VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce rukou v optických obrazech

Hand Detection in Optical Images

2014 Stanislav Velký

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Stanislav Velký**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce rukou v optických obrazech
Hand Detection in Optical Images

Zásady pro vypracování:

Cílem diplomové práce je vytvořit detektor rukou (dlaň a prsty) v optických obrazech. Detektor bude později spolupodílet (nikoli ale v rámci této diplomové práce) na vytváření 3D modelu řidiče ve vozidle. V rámci diplomové práce proveďte následující:

1. Seznamte se s metodami, které lze pro řešení problému použít.
2. Naiplementujte metodu založenou na použití HOG (histogram orientovaných gradientů) a SVM (support vector machines).
3. Vytvořte dostatečně rozsáhlou trémovací množinu.
4. Pokuste se výpočet HOG urychlit pomocí instrukcí AVX, SSE (mějte na paměti možnost použití detektoru v reálném čase).
5. Pečlivě vyhodnoťte úspěšnost metody v podmínkách podobných těm, které mohou nastat v automobilu.

Programové řešení proveďte v C/C++.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Dr. Ing. Eduard Sojka**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 29. dubna 2014 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2014 .....

Velice rád bych poděkoval doc. Dr. Ing. Eduardu Sojkovi za odborné konzultace, které mi pomohly při vypracování této diplomové práce.

Abstrakt

Tato práce se zabývá popisem a implementací metody histogram orientovaných gradientů pro detekci rukou v obraze. Cílem této práce bylo také zrychlit tuto metodu pomocí vektorových instrukcí. Dále je tato práce rozšířena o metody pro sledování objektů. Nejprve je uveden text, který se zabývá popisem jednotlivých metod. Poslední kapitola této práce se zabývá dosaženou úspěšností popisovaných metod.

Klíčová slova: SVM, histogram, Meanshift, Camshift, gradient, vektorové instrukce, SIMD

Abstract

The aim of this thesis is a description and implementation of method histogram of oriented gradients for detection hands in image. The aim of this work was also to speed up this method using vector instructions. In addition this work is extended to include methods for tracking objects. At first is introduced a basic concept of histogram of oriented gradients and tracking objects. Part of this work is a chapter dealing with the success achieved by detecting and tracking hands.

Keywords: SVM, histogram, Meanshift, Camshift, gradient, vector instructions, SIMD

Seznam použitých zkratek a symbolů

SVM	–	Support vector machine
SIMD	–	Single Instruction, Multiple Data
SSE	–	Streaming SIMD Extensions
AVX	–	Advanced Vector Extensions

Obsah

1 Úvod **1**
1.1 Detekce ruky . . . 3

2 Současný stav **5**
2.1 Detektor Viola-Jones . . . 5

3 Histogram orientovaných gradientů **7**
3.1 Příznaky . . . 7
3.2 Popis meotdy Hog . . . 9
3.3 SVM . . . 15

4 Sledování objektů **20**
4.1 Současný stav . . . 20
4.2 Histogramy . . . 21
4.3 Momenty . . . 22
4.4 Meanshift . . . 23
4.5 Camshift . . . 25

5 SIMD **27**
5.1 Flynnova klasifikace . . . 27
5.2 Vývoj SIMD . . . 29

6 Implementace **33**
6.1 Implementace HOG . . . 33
6.2 Implementace MeanShift a CamShift . . . 35

7 Testy a experimenty **37**
7.1 Hog . . . 37
7.2 Meanshift a Camshift . . . 42
7.3 SSE a AVX instrukce . . . 48

8 Závěr **53**

9 Reference **54**

Přílohy **55**

A Diagram tříd **56**

Seznam obrázků

1 Detekce aorty na snímcích z CT [1] . . . 2
2 Detekce defektů na výkovku [2] . . . 2
3 Kinect [3], Asus Xtion [4], Leap Motion [5] . . . 4
4 Rozpoznáváni ruky [6] . . . 4
5 Haarovy příznaky . . . 6
6 Kaskádová klasifikace . . . 6
7 Binární klasifikátor . . . 8
8 Směr hrany vzhledem k směru gradientu . . . 10
9 Hrana se skokovým profilem . . . 11
10 Hrana se střechovým profilem . . . 11
11 Hrana s linkovým profilem . . . 11
12 Hrana se zašumněným profilem . . . 11
13 Schéma HOG . . . 12
14 Zvýrazněné hrany pomocí normalizace obrazu . . . 13
15 Výsledek aplikace filtru . . . 13
16 Rozdělení detekčního okna na mřížky a bloky . . . 14
17 Ukázka rozdělení bloku, normalizace, sloučení histogramů . . . 14
18 Sloučení vzájemně překrývajících oken do jednoho okna . . . 15
19 Ukázka optimální oddělovací nadroviny . . . 16
20 Ukázka mapování z dvou-dimensionálního do tři-dimensionálního prostoru 17
21 Princip konvergence k maximální hustotě . . . 23
22 Ukázka zpětné projekce . . . 25
23 Princip SISD instrukce . . . 28
24 Princip MIMD instrukce . . . 28
25 Princip SIMD instrukce . . . 29
26 Ukázka zrychlení pomocí MMX instrukcí na různých příkladech . . . 31
27 Princip MULPS instrukce . . . 31
28 Princip uložení informací do třírozměrného pole . . . 34
29 Ukázky positivní testovací sady první kolekce . . . 38
30 Ukázky negativní testovací sady první kolekce . . . 39
31 Ukázky positivní testovací sady druhé kolekce . . . 39
32 Ukázky negativní testovací sady druhé kolekce . . . 39
33 Ukázka prostředí ve kterém byly jednotlivé videa natočena, horní obrázek zachycuje první (jednodušší) video, dolní zachycuje ukázku druhého (složitějšího) videa . . . 43
34 Graf nalezené pozice v ose X pro MeanShift . . . 44
35 Graf nalezené pozice v ose Y pro MeanShift . . . 44
36 Graf nalezené pozice v ose X pro CamShift . . . 45
37 Graf nalezené pozice v ose Y pro CamShift . . . 45
38 Graf nalezené pozice v ose X pro MeanShift . . . 46
39 Graf nalezené pozice v ose Y pro MeanShift . . . 46
40 Graf nalezené pozice v ose X pro CamShift . . . 47

41 Graf nalezené pozice v ose Y pro CamShift 47
42 Graf urychlení pomocí SIMD instrukcí a loop unrolling pro operaci skalárního součinu 50
43 Graf urychlení výpočtu příznaků pomocí AVX a SSE instrukcí 51
44 Celkové urychlení výpočtu příznaků a klasifikace pomocí AVX a SSE instrukcí 52
45 Diagram tříd implementace 56

Seznam výpisů zdrojového kódu

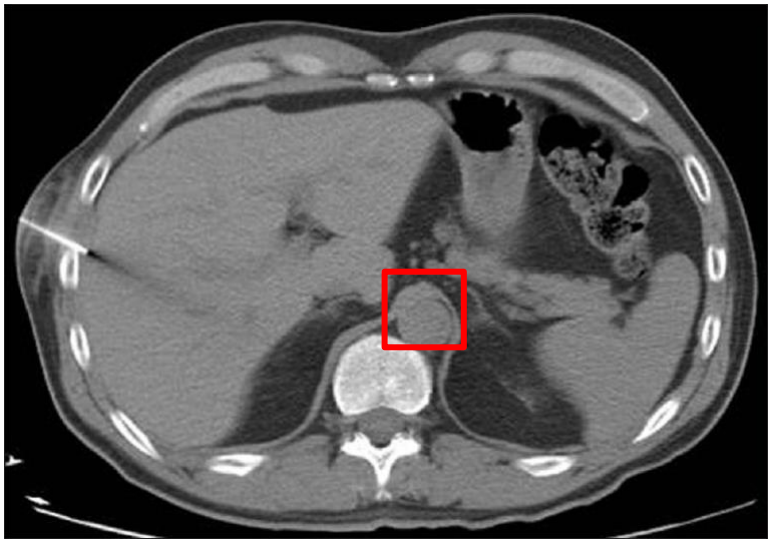
1 Část výpisu skalárního součinu. Na levé půlce jsou instrukce vygenerované v debug režimu na pravé v release režimu 32
2 Pseudokód algoritmu Meanshift 36
3 Pseudokód algoritmu Camshift 36
4 Loop unrolling pro skalární součin 49
5 Kód vygenerovaný kompilátorem pro hlavní smyčku skalárního součinu u instrukcí SSE v release režimu 49
6 Kód vygenerovaný kompilátorem pro hlavní smyčku skalárního součinu u instrukcí AVX v release režimu 49

1 Úvod

Obor počítačového vidění se vyvinul v 60. letech 20. století práci vědce Larryho Robertse, která se zabývala extrahováním 3D informací z 2D perspektivních pohledů. Průlom v této obslati nastal v roce 1978, kdy David Marr přinesl nové přístupy k porozumění scény a tím uvedl nové možnosti k získání 3D scény. Mnozí vědci si však později uvědomili, že k získání informací z 2D scény není nutné vytvářet přímo 3D modely objektů, ale stačí znát pouze 2D informace.

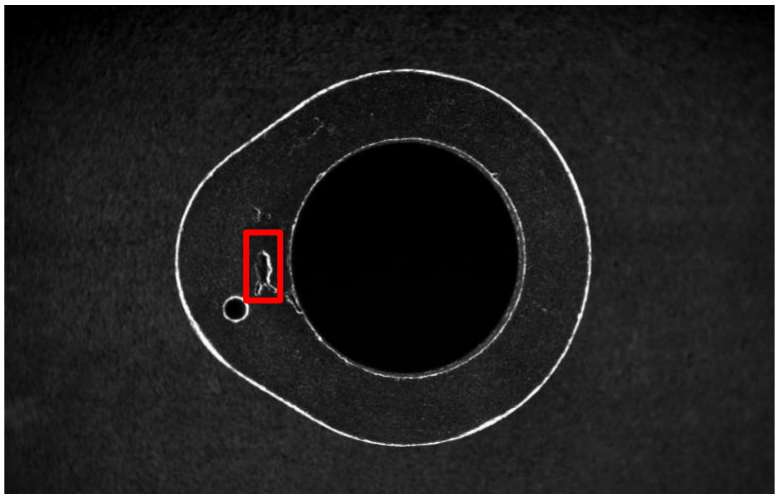
Obor počítačového vidění je z mnoha pohledů velice obtížný. Téměř žádný výzkum v oblasti počítačového vidění nebyl zcela vyřešen. Na první pohled by se mohlo zdát, že detekce obličejů je jeden z problému, kde byly odhaleny všechny problémy. Existuje však mnoho různých obrazů s nevhodným natočením obličeje nebo špatným osvětlením, kde dosud publikované metody nedokázaly obličej nalézt. Mnoho výzkumu se proto zaměřuje na chování lidského mozku a otázku, jak lidský mozek dokáže rozpoznat daný objekt. Z pohledu člověka je rozpoznávání objektů přirozená věc, dokážou na fotografiích rozeznat své přátelé, určit místo pořízení fotografie atd.. Přenesení tohoto problému do strojových instrukcí však přináší nemalé potíže. Lidský mozek je podle mnoha výzkumu mnohem výkonnější než počítač. Dokáže za sekundu provést mnoha trilionů operací a uložit několik tisíc terabajtů informací, což je mnohem více než nejlepší současný superpočítač. Hlavní problém však nastává v otázce, jakým způsobem reprezentovat danou fotografii a jak se tyto informace naučit používat tak, aby výsledná detekce by byla pro počítač co nejjednodušší.

Využití strojového vidění je dnes velice rozšířené a využívá se v mnoha oborech. Zdravotnictví je jeden z oborů, kde počítačové vidění je využíváno nejvíce. Používá se ke stanovení diagnózy pacientů z obrazových dat. Tato data můžou být získané z mikroskopů, rentgenového záření nebo tomografie. Příkladem je detekce nádorů, získávání dat o velikostech orgánů atd..



Obrázek 1: Detekce aorty na snímcích z CT [1]

Využít strojové vidění lze také v průmyslu, kde je součástí výrobního procesu. Podílí se na kontrole výrobků, zda neobsahují nějaké defekty nebo pro získání informací z okolního prostředí pro orientaci robota.



Obrázek 2: Detekce defektů na výkovku [2]

1.1 Detekce ruky

Rozpoznávání objektů se v posledních letech stalo objektem zájmů mnoha prací a mnoha výzkumů založených na hledání nových metod. Nejčastějším objektem pro detekci se v posledních letech staly detekce obličejů, které se využívají v mnoha aplikacích od rozpoznávání identity až po označování osob na sociálních sítích. Další možnosti detekce části těla, která se také stala velmi rozšířenou v počítačovém vidění, je ruka. Detekce rukou může přinést v mnoha oborech velký přínos. Od využití v zábavním průmyslu, kde hlavní význam této detekce je v ovládání her až po využití v rozpoznávání znakové řeči. Samotná detekce rukou v mnoha případech nestačí a je nutné přinést mnoho rozšíření pro reálné uplatnění v praxi. V jíž zmíněné znakové řeči je nutné pro správné rozpoznávání jazyka uplatnit mnoho technik. Od nalezení rukou, přes sledování trajektorie, až po vyhodnocení jejich tvarů. Již první krok, který je hlavním bodem této práce, přináší mnoha úskalí. V porovnání s obličejem, kde můžeme nalézt důležité prvky v detekci například oči a poměrně stálý tvar, tak ruka žádné takové významné prvky nenabízí a může nabývat mnoha tvarů, které detekci znesnadňuji. V posledních letech vzniká mnoho zařízení, která detekci a sledování rukou výrazně ulehčuje. Mezi takovéto zařízení můžeme zařadit kinect, který kromě běžné rgb kamery obsahuje také hloubkový sensor a výrazně tak přispívá ke zjednodušení detekce. Na podobném principu pracuje také zařízení ASUS Xtion. Oba tyto nástroje dokážou kromě detekce a sledování ruky, detekovat osobu do vzdálenosti 5 metrů. Leap Motion je zařízení, které je založené pouze na sledování pohybu ruky a pohyby prstů. Princip Leap Motion je odlišný od výše zmíněných nástrojů. Obsahuje tři infračervené led diody a dvě infračervené kamery, které dokážou zachytit odražené data a poslat na zpracování přes USB do počítače. Toto zařízení dokáže snímat pohyby rukou pouze do vzdálenosti 1 metru.

Obrázek 3: Kinect [3], Asus Xtion [4], Leap Motion [5]

Obrázek 4: Rozpoznáváni ruky [6]

2 Současný stav

Existuje mnoho prací, které zkoumají různé techniky detekce rukou nebo jiných částí těla. Některé jsou založené na zkoumání barvy jednotlivých bodů obrazů. Tyto metody jsou založeny na sestavení barevného modelu ruky a odečtením takových oblastí obrazu, které do tohoto modelu nespadají. Tyto metody jsou závislé na jednotné barvě rukou. Ruce, které jsou ovlivněny osvětlením nebo různými stíny, způsobují velmi nepřesné detekce. Mnoho prací je proto zaměřeno na vytváření takových metod, které jsou invariantní vůči osvětlení nebo stínům. Práce [7] popisuje vytvoření modelu pomocí Bayesovy rozhodovací teorie. Další práce [8] popisuje segmentaci ruky pomocí samoorganizující se mapy (SOM), které jsou založeny na neuronových sítích. Další metody mohou být založené na nalezení kontur v obraze a vyhledání takových, které se nejvíce podobají rukám. Metoda pro detekci ruky, která je založena na podobném principu jako metoda Hog, je Viola Jones. Tato metoda bude zde popsána z důvodu již zmíněné podobnosti s metodou Hog více než metody, které jsou založeny na segmentaci obrazu. Z důvodu vysoké variability lidských rukou a různorodosti prostředí, ve kterém si lidská ruka může nacházet (pozadí obrazu, světelné podmínky), nelze zvolit jedinou metodu, která by obstála ve všech případech. Z tohoto důvodu je proto vhodné využít pro robustnost výsledné detekce kombinace výše zmíněných technik. Taková detekce je však velmi výpočetně náročná a nelze ji tedy využívat v real-time aplikacích.

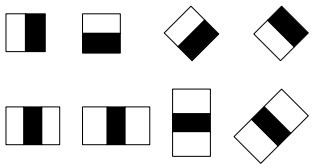
2.1 Detektor Viola-Jones

Metoda Viola Jones [9] byla vyvinuta v roce 2001 autory Paul Viola a Michael Jones. Tato metoda je primárně určena k detekci obličeje, ale lze ji využít k detekci různých objektů. Stejně jako metoda Hog je založena na získávání příznaků a jejich následnou klasifikací. Na rozdíl od metody Hog, kde se jako příznaky používají histogramy orientovaných gradientů, tak zde se používají Haarovy příznaky. Získání příznaků je u této metody časově a výpočetně mnohem méně časově náročnější i díky výpočtu integrálního obrazu. Hodnota příznaků je určena jako rozdíl mezi sumou pixelů pod černou a bílou oblasti. Možné tvary příznaků lze vidět na obrázku (5). Pro urychlení výpočtu těchto příznaků se využívá technika integrálního obrazu. Integrální obraz je definován jako součet předchozích pixelů doleva a nahoru od právě počítaného bodu obrazu. Pixel dole vpravo je tedy součtem všech pixelů v obrazu. Výpočet integrálního obrazu je dán vztahem

$$I(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{1}$$

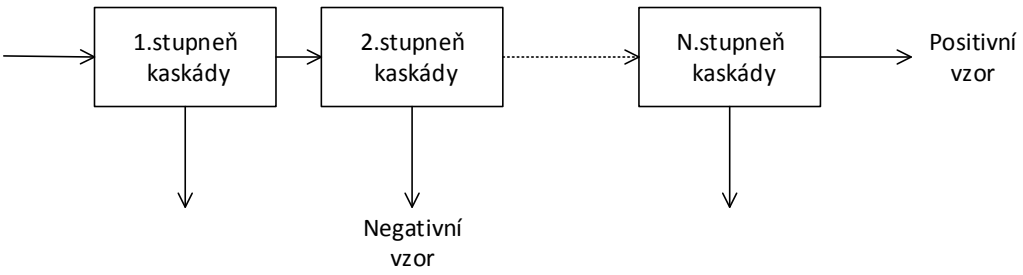
kde $I(x,y)$ a $i(x',y')$ označují hodnoty jednotlivých pixelů. Pomocí takto získaného integrálního obrazu lze vypočítat hodnotu jedné oblasti příznaku jen pomocí čtyř přístupů do paměti.

Dalším rozdílem proti metodě Hog je využívání jiného algoritmu pro klasifikaci příznaků. Pro trénování příznaků se v metodě Viola Jones používá algoritmus AdaBoost. Tento algoritmus byl představen roku 1996 autory Freund a Schapire. Pro svou funkci AdaBoost využívá množinu tzv. slabých klasifikátorů. Za slabý klasifikátor lze označit



Obrázek 5: Haarovy příznaky

metody jako rozhodovací strom nebo perceptron. První klasifikátor má úspěšnost odhadu kolem 50%. Postupným přidáváním nových klasifikátoru získáváme účinnější klasifikaci, která obsahuje méně chybových detekcí. Výsledný klasifikátor je pak dán váženou sumou těchto slabých klasifikátorů. Důležitou podmínkou těchto slabých klasifikátoru je, aby chybovost slabého klasifikátoru byla menší než 0,5.



Obrázek 6: Kaskádová klasifikace

Viola Jones pro hledání objektů využívá tzv. kaskadovovou klasifikaci (6), která má za úkol snížit časovou náročnost klasifikace daného pod-okna. Výsledná klasifikace je pak složena z několika stupňů kaskády. Hlavní myšlenka této klasifikace je ve vybrání takových příznaků do dané části kaskády, které dokážou odhalit negativní okno už na začátku celé kaskády. Pod-okno, které se dostane až na konec této kaskády, lze označit za positivní a odpovídá hledanému objektu.

3 Histogram orientovaných gradientů

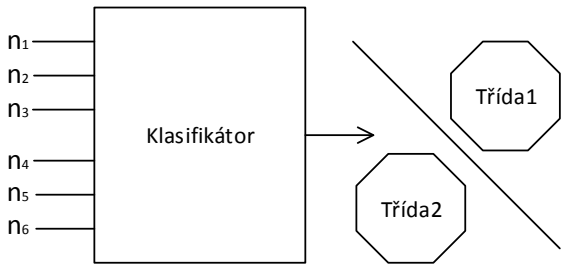
Zadání této práce mi ukládalo implementovat metodu Hog v jazyce C++ pro detekci rukou, z tohoto důvodu bude tato kapitola věnována teoretickým poznatkům, které jsou pro samotnou implementaci důležité. Kapitola se skládá z několika částí popisující extrakci příznaků z obrázku pro detekci rukou. Nejprve bude uvedena podkapitola, která se zabývá obecným popisem příznaků, příklady různých příznaků, které lze využívat pro detekci různých objektů a vhodnosti zvolených příznaků prezentovanými pomocí vztahů, které však nebudou ve výsledné aplikaci použity. Dále bude představena samostatná podkapitola, popisující metodu Hog [10]. Tato kapitola je implementována ve výsledné aplikaci a výsledky této implementace jsou prezentovány v kapitole testy a experimenty, kde je porovnána také s implementací metody Hog z OpenCV. Spolu s touto podkapitolou bude ukázána také metoda pro detekci hran, ve které budou uvedeny některé vztahy důležité pro implementaci metody Hog. Dále bude také představen samostatný algoritmus pro extrakci příznaků metody Hog. Poslední podkapitolou uvedenou v této kapitole je princip trénování a klasifikace získaných příznaků pomocí metody SVM (Support vector machine).

3.1 Příznaky

Důležitým prvkem zpracování obrazu jsou příznaky. Příznaky jsou extrahovány z obrazu za účelem získat z obrazu důležité informace. Existuje celé spektrum příznaků a hlavním cílem je vybrat takové, které hledaný objekt popíši co nejlépe. Příznaky, které popisují daný objekt, pak spadají do jedné třídy a nabývají podobných hodnot. Naopak hodnoty příznaků, které daný objekt nepopisují, musí být co nejvíce vzdáleny od positivních příznaků. Čím více dokážeme toto pravidlo dodržet, tím lépe dokáže klasifikátor oddělit v případě binárního dělení dvě třídy a umožní tak získat lepší detekci objektů. Klasifikátor je algoritmus, který provádí klasifikaci vstupních příznaků. Můžeme je rozdělit na lineární a nelineární. Binární lineární klasifikátory na vstupu očekávají data u nichž lze nalézt nadrovinu, která tato data oddělí. Naopak u vstupních dat, které nejsou lineárně oddělitelné je nutné nalézt oddělující křivku. Klasifikátor tedy na vstupu přijímá N vektorů (příznaků)(x_0 - x_n). Klasifikačním algoritmem je nalezena ze vstupních vektorů (x_0 - x_n) oddělující nadrovina (funkce $d(x)$), podle které lze na výstupu klasifikátoru zjistit identifikátor třídy vstupních příznaků. Pro nalezení rozhodovací funkce $d(x)$ je tedy nutné sestavit algoritmus, který s minimální chybou provede zobrazení vektoru x_i na hledanou třídu. V této práci bude výsledná funkce určena pomocí metody SVM (support vector machine).

3.1.1 Vhodnost zvolených příznaku

Jak již bylo zmíněno v předchozí podkapitole, je nutné volit příznaky, které nejlépe reprezentují danou třídu. Vhodnost těchto příznaku lze ověřit pomocí matematických vztahů. Pro ukázku výpočtu vhodnosti těchto příznaku jsou zvoleny pouze dva příznaky (x, y). Uvedené vztahy zjistí vhodnost jednotlivých prvků vektoru příznaků. Pokud budou



Obrázek 7: Binární klasifikátor

podle uvedených vztahů nevhodné nebo nadbytečné lze je z uvedeného vektoru příznaků vypustit. Vhodnost těchto příznaků lze zjistit pomocí normalizované kovariance a vzdálenosti mezi třídami. Pro výpočet je nejprve nutné vypočítat střední hodnotu a varianci těchto příznaků v daných třídách [11].

$$\mu_{x,i} = \frac{1}{N_i} \sum_{j=1}^{N_i} x_{i,j} \tag{2}$$

$$\sigma_{x,i}^2 = \frac{1}{N_i} \sum_{j=1}^{N_i} (x_{i,j} - \mu_{x,i})^2 \tag{3}$$

kde Ni je počet hodnot v jednotlivých třídách. Z předchozího textu víme, že hodnoty dané třídy musí si být co nejvíce podobné, proto hodnota variance by měla nabývat co nejmenší hodnoty. Normalizovaná kovariance je definována vztahem

$$\sigma_{xy,i}^2 = \frac{1}{N_i \sigma_{x,i} \sigma_{y,i}} \sum_{j=1}^{N_i} (x_{i,j} - \mu_{x,i})(y_{i,j} - \mu_{y,i}) \tag{4}$$

a udává závislost příznaků. Hodnota 0 určuje nezávislost. Absolutní hodnota kovariance větší jak 0 určuje závislost. Hodnota kovariance 1 značí úplnou závislost a je tedy vhodné z důvodu nadbytečnosti jeden z příznaků x, y vypustit. Dalším kritériem je vzdálenost mezi třídami, která je definována vztahem

$$D_{x,i,j} = \frac{|\mu_{x,i} - \mu_{x,j}|}{\sqrt{\sigma_{x,i}^2 + \sigma_{x,j}^2}} \tag{5}$$

z předchozího textu je důležité, aby výsledná hodnota byla co největší a tedy výsledná klasifikace by byla co nejpřesnější.

Mezi další vlastnosti, které jsou pro výslednou klasifikaci podstatné, lze zařadit.

1. Rychlost výpočtu – je vhodné, aby výsledný čas k získání příznaku netrval příliš dlouho. Avšak vyšší časová náročnost ve většině případů může znamenat lepší

reprezentaci daného objektu. Za vysokou časovou náročnost můžeme považovat také vysoký počet nebo velkou dimensi vektoru, která zásadně zpomaluje čas v výsledné klasifikaci. Tento problém nastane také v mé práci, kde velká dimense vektoru příznaku a vysoký počet těchto příznaků způsobí zásadní zpomalení celé klasifikace.

2. Invariantnost – výsledná hodnota příznaku nesmí být závislá na změně jasu nebo kontrastu.

Z výše uvedeného textu vyplývá, že vhodný výběr příznaků může znamenat vysokou účinnost pro správné rozpoznávání daného objektu. Jedním ze základních příznaků, které lze využít jsou momenty. Tyto příznaky jsou snadné na výpočet a dokážou účinně od sebe odlišit jednotlivé třídy. Momenty budou také využívány v kapitole pro sledování objektů. Pro rozlišení jednotlivých tříd objektů lze využít také příznak *kruhovost*, která je definována vztahem

$$C = \frac{P^2}{A} \tag{6}$$

, kde P je délka hranice objektu a A jeho plocha. Výsledek vztahu (6) určuje tvar výsledného objektu. Pro kruh je definována hodnota 4π, pro čtverec hodnota 16, pro objekty nepravidelného tvaru jsou hodnoty vyšší. Objekty lze rozpoznávat také podle křivosti jeho hranice nebo podle eulerova čísla. Ve své práci budu k rozlišení objektů (ruky) od jiných objektů využívat příznaky získaných z gradientů obrázku.

3.2 Popis meotdy Hog

Histogram orientovaných gradientů (Hog) metoda je založena na získávání příznaků z obrazu v podobě histogramu orientovaných gradientů. Metodu navrhli autoři Navneet Dalal a Bill Triggs a původně byla určena pro detekci osob. Základním prvkem této metody je výpočet gradientu, tedy vektorové funkce, určující směr a velikost největšího růstu. Získaný směr a velikost gradientů je základním bodem pro vytvoření jednotlivých histogramů. Autoři této metody také navrhli různé metody normalizace pro jednotlivé histogramy, které zde budou dále také popsány. Výpočet příznaků z velké části vychází z metody pro detekci hran (Cannyho detektor hran), proto bych chtěl ukázat princip této metody ve spojení s metodou Hog.

3.2.1 Cannyho detektor hran

Důležitým prvkem ve zpracování obrazu je detekce hran. Na toto téma vzniklo již mnoho metod avšak jako nejrozšířenější lze považovat Cannyho detektor hran.

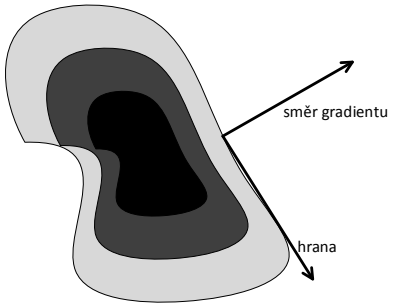
Hrany v obraze vznikají v místě, kde dochází k náhlé změně jasu. Hrana může vznikat na hranici dvou objektů nebo mezi světlem a stínem. Takto definovaná hrana tedy bude v místě náhlé změny jasu vykazovat vysoké hodnoty první derivace. Cannyho detektor využívá tohoto pozorování a pro nalezení hrany provádí derivaci jasové funkce v obraze. Derivaci v případě Cannyho detektoru je nahrazena aproximaci derivace pomocí

tzv. operátorů. Zvolený operátor pomocí matematické operace konvoluce (matematická operace která kombinací dvou signálů tedy obrazu a operátoru vytváří modifikovaný třetí signál) vytváří novou hodnotu bodu v obraze. Operátor je definován maticí, která se pokládá na každý pixel vstupního obrazu a vynásobením hodnot v obraze s hodnotami v operátoru, získáme novou hodnotu. Před samotnou derivací (aplikací konvoluce) lze na obraz aplikovat pro zlepšení detekce hran Gaussový filtr (7), který zamezí vzniku falešných hran rozostřením obrazu, avšak v metodě Hog kvůli větší rychlosti aplikován nebude.

$$g(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{\frac{x^2+y^2}{2\sigma^2}} \tag{7}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{8}$$

Pro aproximace derivace lze využít několik operátorů (Sobel (8), Roberts), které se od sebe liší velikostí nebo jinými hodnotami v matici. Operace konvoluce s těmito operátory je časově náročnější z důvodu většího počtu operace násobení. Proto lze tyto operátory nahradit v metodě Hog filtrem, který je jednodušší (rychlejší) a vykazuje v metodě Hog lepší výsledky (12). Využitím filtrů (12) získáme gradient (9) nebo-li vektor parciálních derivací vyjadřující směr a velikost největší změny funkce. Výsledná hrana je kolmá ke směru gradientu. Na obrázcích (9 - 12) můžeme pozorovat profily, kterých hrana může nabývat. V obraze se však nejvíce setkáváme s profilem na obrázku (12).



Obrázek 8: Směr hrany vzhledem k směru gradientu

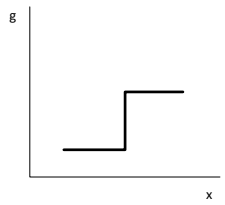
$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \tag{9}$$

Pro získání směru hrany využijeme vztah

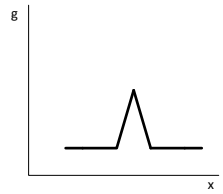
$$\theta = \arctan(\frac{G_y}{G_x}) \tag{10}$$

Velikost (síla) hrany je definována vztahem

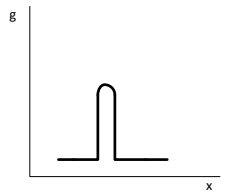
$$G = \sqrt{G_x^2 + G_y^2} \tag{11}$$



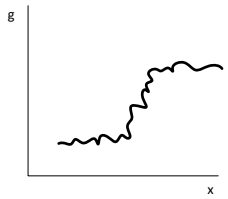
Obrázek 9: Hrana se skokovým profilem



Obrázek 10: Hrana se střechovým profilem



Obrázek 11: Hrana s linkovým profilem



Obrázek 12: Hrana se zašumněným profilem

Pomocí hodnot získaných ze vztahu (11)(10) jsou v metodě Hog sestaveny histogramy, které určují výsledný vektor příznaků. Rozpoznávaný objekt je tedy určen směry a velikostí jednotlivých hran v daných místech obrazu.

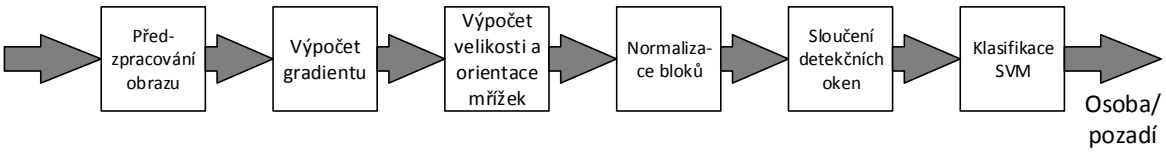
Další kroky, které zde budou popsány se v metodě Hog nevyskytují, avšak pro úplnost Cannyho detektoru je vhodné tyto kroky zmínit. Pomocí konvoluce získáme tedy ze vstupního obrazu tzv. binární obraz, kde větší hodnoty jasu pixelů určují větší velikost hrany, tedy větší změnu jasu. K získání hran je potřeba udělat kroky, které určí výsledné hrany.

1. Ztenčení – v této fázi dochází k odstranění (k vynulování) pixelů, které nejsou maximem (zachovány budou pixely s největším gradientem). Pixely, které zůstanou zachovány jsou nalezeny tak, že hodnoty okolních pixelů jsou ve směru a proti směru gradientu (vztah 10) nižší, než aktuálně zkoumaný.

2. Prahování – dosud je každý pixel s hodnotou větší jak nula označen jako hrana. Pixely s velmi malou hodnotou gradientu však hranu neoznačují. Pro nalezení skutečných hran se zvolí dva prahy P1 a P2. Pixely s hodnotou gradientů menší jak P1 jsou vynulovány (nejedná se o hrany). Pixely vetší jak P2 jsou automaticky označeny za hrany a pixely mezi P1 a P2 jsou označeny jako hrany pouze v případě, že jejich nejbližší soused je označen jako hrana, tedy má hodnotu vetší jak P2.

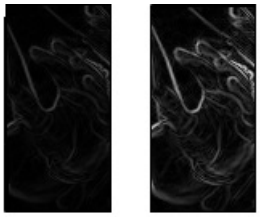
3.2.2 Princip Hog

V předchozí kapitole byly zobrazeny vztahy pro získání velikosti a směru gradientů. Hlavní myšlenou této metody je využít vztahu (10) a (11) a popsat pomocí nich obraz. Pomocí těchto hodnot jsou sestaveny histogramy, které určují výslednou strukturu hledaného objektu (ruky).



Obrázek 13: Schéma HOG

1. Prvním krokem řetězce je předzpracování obrazu. Než bude na vstupním obraze proveden výpočet pro získání příznaků, je nutné provést se vstupním obrazem několik operací. Nejprve je vstupní obraz převeden z R, G, B hodnot do jasového zobrazení. Tento převod umožní vypočítat gradient namísto ze tří složek, ze kterých by byla vybrána hodnota s největším gradientem, pouze z jedné a tím výslednou detekci výrazně urychlit. Před samotným aplikováním konvoluce může být na jasový obraz aplikován normalizační výpočet, který převede jasové hodnoty do rozmezí $0 - 255$ a tím více zvýrazní jednotlivé hrany.



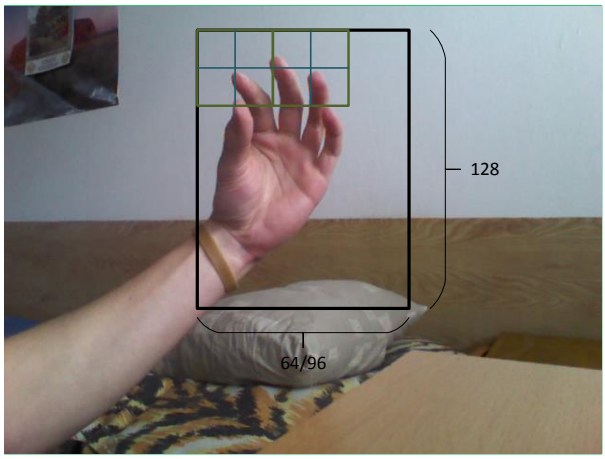
Obrázek 14: Zvýrazněné hrany pomocí normalizace obrazu

2. Po aplikování operací z prvního kroku je možné provést výpočet derivace ve směru osy x a y. Tento výpočet jak bylo zmíněno výše slouží pro určení velikosti a směru gradientu a lze jej provést např. pomocí Sobelova operátoru. Z důvodu vyšší časové náročností tento operátor použit nebude a bude nahrazen jednoduššími výpočty (12). Ve výsledku operátor (12) vykazuje zlepšení a hlavně zrychlení detekce než za použití Sobelova operátoru.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T \tag{12}$$

Obrázek 15: Výsledek aplikace filtru

3. V této fázi rozdělíme detekční okno, které na obrázku (16) nabývá hodnot $(32/64)x128$ pixelů (tyto rozměry budou použity také pro hledání rukou ve výsledném testování), na jednotlivé mřížky a z každé mřížky se vytváří histogram. Velikost mřížek závisí na typu rozpoznávaných objektů. Podle Delal a Triggs optimální velikost mřížky pro rozpoznávání osob je $8x8$ pixelů. V kapitole Testy a experimenty uvidíme, že tato velikost je vhodná také pro rozpoznávání rukou.
   Každá mřížka bude tedy představovat jeden histogram. Histogram je určen směrem a velikostí gradientů podle vztahů (10)(11). Velikost tohoto histogramu je experimentálně určena na základě testování různých vstupních obrazů. Podle Delal a Triggs optimální velikost histogramu mřížky pro rozpoznávání osob je 9 binů. Bin představuje pozici v histogramu, která je dána směrem na základě vztahu (10). Před tím než je určen výsledný bin je potřeba hodnotu směru převést z gradiánu na stupně a normalizovat na rozsah $0 - 360°$. Výsledná pozice v histogramu je pak

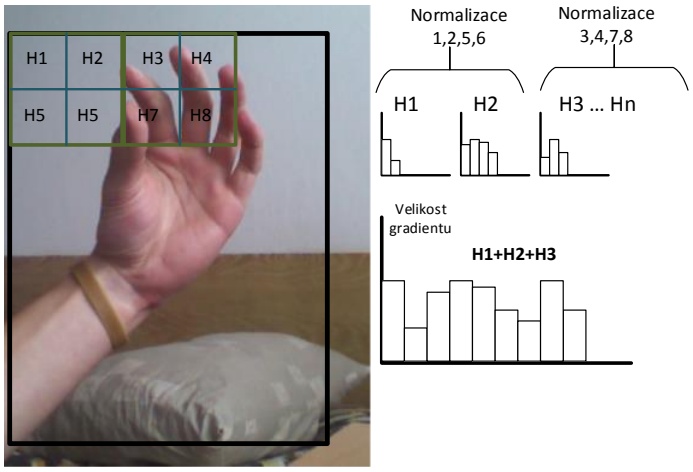


Obrázek 16: Rozdělení detekčního okna na mřížky a bloky

určena vztahem

$$P = (smr\_gradientu\_ve\_stupnch/360) * pocet\_bin \tag{13}$$

Takto vypočtená pozice nemusí vždy nabývat celých hodnot, proto k redukování aliasingu je nutné velikost gradientů interpolovat mezi dva sousední biny. Například pokud hodnota P nabývá hodnoty 2.125 bude k binu 2 přičtena hodnota $velikost\_gradientu * 0.875$ a k binu 3 přičteme velikost $velikost\_gradientu * 0.125$.



Obrázek 17: Ukázka rozdělení bloku, normalizace, sloučení histogramů

4. Vlivem osvětlení a kontrastu se velikost gradientů může lišit, proto se provádí seskupování mřížek do bloků. Každý blok tedy obsahuje několik mřížek, které jsou

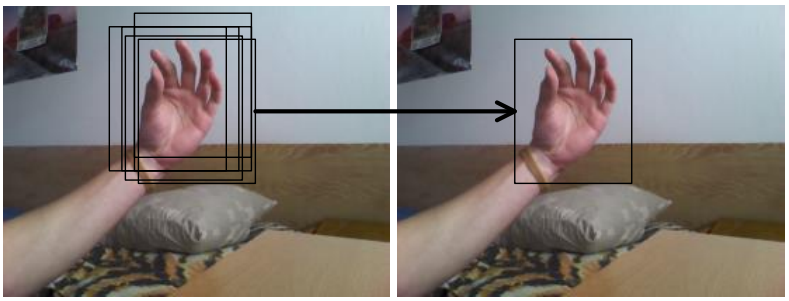
v rámci bloku normalizovány pomocí vztahů (14)(15)(16). Normalizované bloky se také můžou vzájemně překrývat tzn. jedna mřížka bude použita ve více normalizačních bloků. Výsledný počet příznaků u takto překrývajících bloků bude z důvodu vícero využití stejné mřížky větší, což může mít za následek zlepšení detekce, ale také složitější a časově náročnější výpočet. Práce autorů Delal a Triggs popisuje také možnost zvolit namísto čtvercového normalizačního bloku kruhový. Takto provedena normalizace je ovšem časově náročnější a nepřináší žádné výrazné zlepšení, proto v této práci budou testovány pouze čtvercové bloky různé velikosti. Normalizované histogramy budou přes celé detekční okno sloučeny do jednoho histogramu (pole), který udává výsledný vektor příznaků. Získané vektory příznaků budou použity jako vstup pro klasifikátor. V této práci bude klasifikace realizována pomocí metody SVM.

$$f = \frac{v}{(||v||_1 + e)} \tag{14}$$

$$f = \sqrt{\frac{v}{(||v||_1 + e)}} \tag{15}$$

$$f = \frac{v}{\sqrt{(||v||_2^2 + e^2)}} \tag{16}$$

5. V rámci úspěšné detekce objektů se může stát, že algoritmus vrátí několik vzájemně posunutých oken. Takto vzájemně překrývající nalezená detekční okna lze sloučit do jednoho. Výsledné okno bude nabývat pozice a velikosti, které bude zprůměrováno z těchto překrývajících oken.



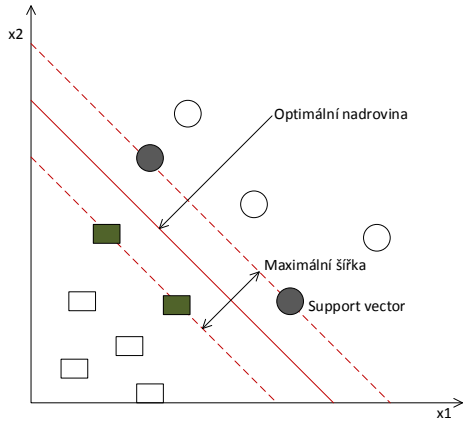
Obrázek 18: Sloučení vzájemně překrývajících oken do jednoho okna

3.3 SVM

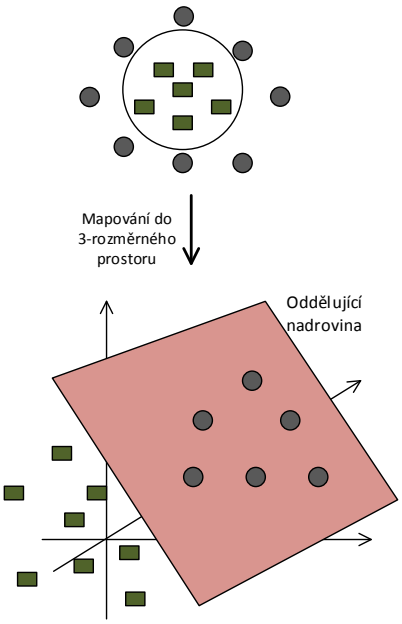
Předchozí kapitola byla věnována extrakci příznaků pomocí metody Hog. Tato kapitola bude věnována využití těchto příznaků v učení a následném rozpoznávaní objektů.

Mnoho algoritmů pro nalezení optimální hranice mezi dvěma třídami jsou velice jednoduché a srozumitelné, avšak tyto algoritmy jsou schopny nalézt optimální separující nadrovinu pouze pro lineárně oddělitelné třídy. Algoritmy, které dokážou oddělit nelineárně separabilní třídy, jsou často obklopeny mnoha problémy. Mezi takovéto algoritmy lze zařadit neuronové sítě. Mezi hlavní problém neuronových sítí a algoritmu backpropagation je nejen obtížné učení, ale také možnost uvíznutí v lokálním minimu. Toto uvíznutí může zapříčinit horší klasifikaci než v případě, že kdyby bylo nalezeno globální minimum. Výsledná klasifikace je také závislá na určení hodnot počátečních vah a nevhodné vygenerování těchto vah může znovu vést k uvíznutí v lokálním minimu.

Metoda SVM [12] patří k novějším metodám, která původně byla určena pouze k binární klasifikaci. Toto omezení však lze překonat, ale z důvodu využití pouze binární klasifikace v rámci práce, tuto možnost zde nebudu ukazovat. Metoda je určena nejen pro lineárně oddělitelné třídy, ale dokáže také reprezentovat složité nelineární funkce a tím umožnit oddělit lineárně neseparabilní třídy. Hlavní myšlenkou lineárního oddělení lineárně neseparabilních tříd je převedení vstupního prostoru na více-dimensionální prostor. Tímto převodem získáme prostor ve kterém je možné lineárně od sebe oddělit dané třídy. Tento převod lze vidět na obrázku (20), kde jsou zobrazeny příznaky, které lze oddělit od sebe do správných tříd pomocí nelineární funkce (kružnice), ale po převodu do vyšší dimenze už existuje nadrovina, která toto oddělení dokáže vytvořit.



Obrázek 19: Ukázka optimální oddělovací nadroviny



Obrázek 20: Ukázka mapování z dvou-dimensionálního do tři-dimensionálního prostoru

Předpokládejme, že jako vstup máme množinu $x_i \in R$ a $y_i \in \{-1, 1\}$, kde x_i vstupní vektor a y_i označuje třídu, do které tento vektor spadá, hodnota i nabývá $i \in < 0, n >$. Cílem algoritmu SVM je najít nadrovinu s maximálním okrajem, která dokáže oddělit třídy $y_i \in \{-1, 1\}$. Nadrovinu oddělující prostor lze zapsat takto

$$w \cdot x - b = 0, \tag{17}$$

kde w označuje normálový vektor a b/w označuje vzdálenost nadroviny od počátku. Pokud máme lineárně oddělitelné třídy, lze najít takové nadroviny, které mezi sebou neobsahují žádné body a jejich vzdálenost je maximální. Nadroviny, které procházejí support vectory lze zapsat vztahy

$$w \cdot x^+ - b = 1 \tag{18}$$

$$w \cdot x^- - b = -1 \tag{19}$$

Výpočtem rovnice

$$\begin{aligned} w \cdot x^+ - b - 1 &= w \cdot x^- - b + 1 \\ ||x^+ - x^-|| &= \frac{2}{||w||} \end{aligned} \tag{20}$$

zjistíme vzdálenost mezi nadrovinami, který nabývá hodnoty $2/w$. Maximalizováním $\frac{2}{||w||}$ nebo minimalizováním $||w||$ dostaneme maximální vzdálenost nadrovin. Minimalizaci $||w||$ můžeme přepsat na minimalizaci

$$\frac{1}{2}||w||^2. \tag{21}$$

Přičemž platí omezení

$$y_i(w \cdot x_i - b) \geq 1 \tag{22}$$

Úlohu 21 lze převést na hledání sedlového bodu Lagrangeovy funkce

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i[y_i(w \cdot x_i - b) - 1], \tag{23}$$

kde a jsou Langrangovy multiplikátory. Cílem je najít minimální w a b (musí platit $a_i \geq 0$). Toho dosáhneme položíme-li první derivaci podle w a b rovno nule.

$$\frac{\partial L(w, b, \alpha)}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \tag{24}$$

$$\frac{\partial L(w, b, \alpha)}{\partial b} = w - \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{25}$$

Dosazením derivací (25) a (24) dostaneme výraz

$$\begin{aligned} L(w, b, \alpha) &= \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i[y_i(w \cdot x_i - b) - 1] \\ &= \frac{1}{2} < w, w > - \sum_{i=1}^{n} \alpha_i y_i(w \cdot x_i - b) + \sum_{i=1}^{n} \alpha_i \\ &= \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j < x_i x_j > - \sum_{i=1}^{n} \alpha_i y_i < w, x_i > + b \sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i \\ &= \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j < x_i x_j > - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j < x_i x_j > + \sum_{i=1}^{n} \alpha_i \\ &= -\frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j < x_i x_j > + \sum_{i=1}^{n} \alpha_i \end{aligned} \tag{26}$$

Z výrazu (26) získáme hodnoty složek a_i. Datové body, u kterých tyto složky nabývají nenulové hodnoty, jsou označeny jako *support vectory* a leží nejblíže oddělující nadroviny. Důležitou vlastnosti SVM je nalézt co nejvíce složek s hodnotou nula, tak aby efektivita výsledného klasifikování byla co nejmenší. Výsledná klasifikace nabývá tvaru

$$f(x) = sign\left(\sum_{i=1}^{n} \alpha_i y_i (x * x_i)\right) \tag{27}$$

Ve výrazu (26) lze nahradit $< x_i x_j >$ tvarem $k < x_i x_j >$, který se nazývá jádrová funkce. Pomocí tohoto přepisu lze najít lineární oddělovač u nelineárně oddělitelných tříd ve vícerozměrném prostoru. Správný výběr funkce závisí na řešeném problému. V této práci bylo experimentováno s funkcemi (28) a (30), avšak z důvodu téměř totožné účinnosti byla pro výsledné testování určena funkce (28). Využívaná jádrová funkce musí být spojitá, symetrická. Příkladem jádrových funkcí jsou [13]

$$k(x, y) = x^T y + c \tag{28}$$

$$k(x, y) = (\alpha x^T y + c)^d \tag{29}$$

$$k(x, y) = exp(\gamma ||x - y||^2) \tag{30}$$

$$k(x, y) = exp(-\frac{||x - y||}{2\sigma^2}) \tag{31}$$

4 Sledování objektů

V rámci této diplomové práce bude také prezentováno téma sledování objektu (ruky), které není obsaženo v zadání práce, avšak k tématu nalezení ruky se vztahuje. Kapitola je rozdělena na několik částí. První část se věnuje obecným popisem metod, které lze ke sledování objektů využívat. Druhá část histogramy stejně jako třetí zbývající se momenty prezentuje základní vztahy důležité pro vlastní implementaci metod MeanShift a Camshift. Poslední dvě podkapitoly jsou věnovány samotným popisem algoritmů pro sledování objektů MeanShift a Camshift. Výsledky implementace těchto metod jsou prezentovány v kapitole Testy a experimenty.

4.1 Současný stav

Sledování objektů lze označit jako proces, při kterém dochází k lokalizaci objektu v závislosti na čase. Využití tohoto procesu lze najít v mnoha oborech. Od možnosti využití v zábavním průmyslu, kde v závislosti na pohybu různých části těla dochází k interakci s elektronikou přes bezpečnost až po sledování dopravy.

Existuje spoustu metod pro sledování objektů a lze je rozdělit do několika skupin podle typu sledování na siluetové, bodové a jádrové.

- *Siluetové* [14] metody pracují na základě modelu, který je reprezentován pomocí hran objektů získaných pomocí některé z technik pro detekce hran nebo získaného obrysu sledovaného objektu. Sledování pak probíhá na základě porovnávání modelu (například model hran) získaného v předchozím snímku s možnými modely objektu v aktuálním obraze. Podle míry podobnosti je pak vyhodnocen nejlepší objekt v aktuálním snímku a aktuální model je nahrazen novým modelem nalezeného objektu.

- *Bodové sledování* [15] probíhá na základě nalezení významných bodů popisující objekt a korespondenci bodů v následujícím snímku. Nalezení korespondence bodů pak může probíhat na základě různých omezujících podmínek, kdy je korespondence bodů určena na základě znalostí daného problému. Mezi takové podmínky můžeme zařadit předpoklad velmi malé změny polohy bodu v navazujících snímcích, určení maximální vzdálenosti pohybu bodů a tím prohledávat pouze část obrazu nebo vyhledávání bodů v předpokládaném směru a rychlosti. Mezi další možné metody korespondence bodů lze zařadit Kalmanův filtr. Predikce polohy bodů je pak určena pomocí statistického měření.

- Pro mou práci jsem vybral sledování objektů pomocí tzv. *jádrových* metod. Tyto metody pracují pomocí modelu vytvořeného na základě nalezeného objektu. Za reprezentaci modelu lze zvolit histogram, který bude využit i v této práci. Polohu nové oblasti objektu pak lze určit na základě míry podobností modelu. V této práci uvedu dvě sobě podobné techniky, které jsou založeny právě na jádrovém sledování objektů. První algoritmus je nazván Meanshift prezentován v roce 1975

autory Fukunaga a Hostetler. Druhý algoritmus, který rozšiřuje možnosti prve uvedeného, se nazývá Camshift.

Poslední metodu, kterou bych zde rád zmínil, je sledování objektů pomocí odečítání pozadí. Nutný předpoklad pro správnou funkci této metody je statická kamera (nesmí být ovlivněna žádnými otřesy) bez artefaktů vznikajících v obraze. Činnost algoritmu je závislá na získání modelu (obrázku pouze statických částí) daného prostředí. Model je možné získat mnoha způsoby. Nejjednodušší způsobem pro získání modelu je zprůměrováním několika po sobě jdoucích snímků tak, aby byly co nejvíce eliminovány měnící se části obrazu (například projíždějící auta). Kromě průměrování lze zvolit také výpočet mediánu. Z důvodu časté změny počasí například u sledování dopravy je nutné tyto modely vytvářet v určitých intervalech, aby byla odstraněna možnost vzniku stínů nebo eliminace měnící se intenzity osvětlení. Určení výsledných, měnících se části obrazu lze určit pomocí vztahu

$$f(x,y) = \begin{cases} 1 & \text{if } |I(x,y) - B(x,y)| > p \\ 0 & \text{else} \end{cases}$$

kde f(x,y) představuje výslednou hodnotu pixelu tedy obrázek, kde jsou barevně od sebe odlišeny pohybující se objekty od statických (pozadí), I(x,y) udává intenzitu pixelu v aktuálním snímku a B(x,y) hodnotu pixelu získaného z modelu. Prah p je volen experimentálně. Výsledný obrázek může trpět mnoha artefakty. Od označení malých shluků pixelů, které jsou označeny jako popředí, i když většinou se jedná o pouhý šum, tak po nesouvislé označené pohybující se objekty. Tyto artefakty lze do jisté míry minimalizovat pomocí morfologických operátorů (eroze, dilatace, otevření, uzavření ...), kde šum může být odstraněn pomocí eroze nebo nesouvislé objekty lze upravit pomocí dilatace. Jak už bylo uvedeno výše tato technika může trpět měnícím se pozadím a jíž zmíněným artefaktům. Metoda Mixtura gaussiánů se snaží takovým chybám předcházet a vytvářet modely na základě distribučních funkcí normálního rozdělení. Přesný popis algoritmu lze najít zde [16].

4.2 Histogramy

Pro reprezentaci obrazu nemusí být vždy vhodné používat samotné pixely obrazu. K reprezentování obrazu lze využívat mnoho technik, které daný obraz budou reprezentovat mnohem lépe a umožní tak využívat různé vlastnosti obrazu. Mezi takové techniky můžeme zařadit histogramy, které poskytují informaci o počtu jednotlivých barev (jasů). Histogram lze zobrazit v podobě sloupcového grafu se sloupci stejné šířky, které vyjadřují pozici dané barvy a výška sloupců určuje četnost těchto barev. Pro vytvoření histogramu jedno-kanálového obrazu je potřeba zjistit hodnotu daného pixelu. Tato hodnota určuje pozici na ose x daného histogramu. Osa y dané pozice bude inkrementována o určitou hodnotu.

Výpočet histogramu pro barevné obrázky je komplikovanější. Nejprve je nutné provést normalizaci (32) jednotlivých složek pixelu. Tato normalizace umožní zmenšení ve-

likosti daného histogramu. Hodnota N ve vztahu (32) určuje maximální hodnotu daného kanálu.

$$\begin{aligned} k_R &= R * N/M \\ k_G &= G * N/M \\ k_B &= B * N/M, \end{aligned} \tag{32}$$

kde M je maximální hodnota složek R, G, B. Výslednou pozici v histogramu určíme pomocí vztahu

$$Pos = k_R + k_G * N + k_R * N^2 \tag{33}$$

4.2.1 Porovnávání histogramů

Důležitou vlastnosti histogramu je jejich vzájemné porovnávání. Existuje mnoho technik ke zjištění míry podobnosti histogramů. Jednou z nich je pomocí využití Bhattacharyya koeficientu

$$\rho = \sum_{u=1}^{m} \sqrt{p_u q_u} \tag{34}$$

kde p a q jsou porovnávané histogramy. Výsledek Bhattacharyya koeficientu nabývá hodnot $0 - 1$, kde hodnota 1 značí identické histogramy. Výpočet lze aplikovat pouze na normalizované histogramy se shodnou velikostí N v jednotlivých barevných složkách.

4.3 Momenty

Důležitou úlohu v algoritmu Menashift mají momenty. Momenty označují matematické funkce, které slouží pro popis objektů v obraze. Obecná definice momentu lze vyjádřit vztahem

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y) \tag{35}$$

V následujících algoritmech budou konkrétně využívány tyto momenty

$$M_{00} = \sum_x \sum_y I(x, y) \tag{36}$$

$$M_{10} = \sum_x \sum_y xI(x, y) \tag{37}$$

$$M_{01} = \sum_x \sum_y yI(x, y) \tag{38}$$

$$M_{11} = \sum_x \sum_y xyI(x, y) \tag{39}$$

$$M_{20} = \sum_x \sum_y x^2 I(x,y) \tag{40}$$

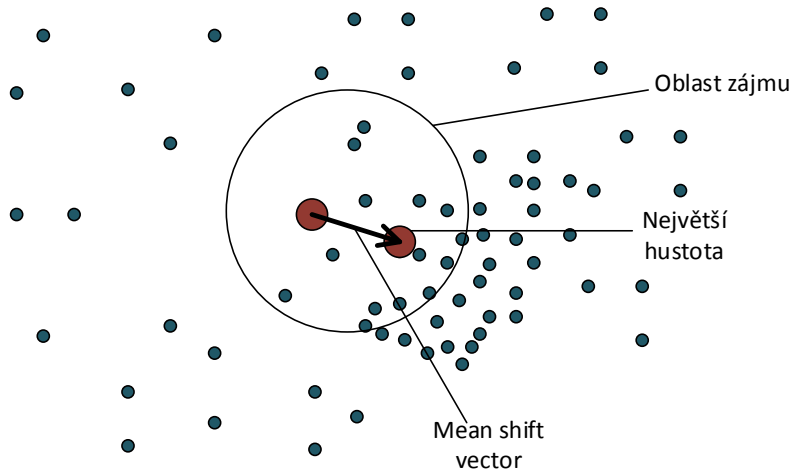
$$M_{02} = \sum_x \sum_y y^2 I(x,y) \tag{41}$$

4.4 Meanshift

Meanshift [17] je technika, která neslouží jen ke sledování objektů v obraze, ale hlavní úkol tohoto algoritmu spočívá v segmentaci obrazu. Meanshift algoritmus je založen na minimalizování vzdáleností funkcí hustoty pravděpodobností mezi cílovým a kandidátním histogramem. Vzhledem k tomu, že tato technika je založena na barevném histogramu a nezajímá ji struktura daného objektu, tak lze ji využívat u objektů, které často mění svůj tvar. Pro objekty nebo prostředí, ve kterém dochází k velké změně barevných složek obrazu vlivem osvětlení nebo stínu, tato technika není vhodná a lze zvolit jiný způsob sledování například pomocí již zmiňovaného siluetového sledování.

4.4.1 Princip

Princip algoritmu Meanshift spočívá v nalezení lokálního maxima hustoty vzorků (pixelů). Algoritmus pracuje iterativně tzn. v každém kroku spočítá vážený průměr všech pixelů v okně a na základě histogramů (cílového a kandidátního) zjistí směr dalšího posunu. Tyto kroky provádí tak dlouho dokud nedosáhne lokálního maxima hustoty pixelů nebo nepřekročí povolený počet iterací.



Obrázek 21: Princip konvergence k maximální hustotě

4.4.2 Algorimus Meanshift pro sledování objeků

Základní krok algoritmus je výpočet cílového histogramu (p) sledované oblasti, která byla nalezena pomocí libovolné techniky detekce objektů. Pro dosažení lepších výsledků je v algoritmu využíván výpočet pro barevný histogram. Vypočítaný cílový histogram bude udržován v paměti po celou dobu sledování objektů. Dále je potřeba uchovávat pozici a velikost posledního nalezeného okna. Druhým krokem je nalezení nového kandidátního histogramu (q) z nově příchozího obrázku. V tomto kroku je předpokládáno, že pozice objektu bude vůči předchozímu snímku posunuta. Okno pro výpočet kandidátního histogramu však bude umístěno na stejné pozici jak v předchozím kroku. V této fázi bude probíhat konvergence okna na nejlepší možnou pozici (nalezení lokálního maxima hustoty vzorků). Nalezení této pozice je dáno určením váhy každého bodu (x, y) v okně. Váha je dána vztahem

$$w_i = \sqrt{\frac{p_u}{q_u}}, \tag{42}$$

kde index u určuje pozici v histogramu, který je získán na základě aktuální hodnoty pixelu. Získanou hodnotu ze vztahu (42) dosadíme za hodnotu $I(x, y)$ do vztahů pro výpočet momentů M_{10} (37) a M_{01}(38). Získané hodnoty momentů ze vztahů (37) a (38) budou vyděleny součty všech vah v okně moment M_{00} (36). Odečtením vypočtené hodnoty (x_c, y_c) od středu výpočetního okna získáme *Meanshift* vektor, tedy vektor udávající směr a velikost s největší hustotou vzorků. Posunem výpočetního okna o daný vektor získáme novou pozici okna, na kterou bude toto okno přesunuto. Celá tato procedura konvergence se bude opakovat tak dlouho, dokud velikost vektoru nebude pod určitou hodnotou nebo nebude překročen počet iterací.

4.4.3 Vylepšení algoritmu Meanshift

Algoritmus popsaný v předchozím kroku má mnoho nedostatků. Sledované objekty většinou nabývají různých tvarů. Vzhledem k obdélníkovému tvaru sledované oblasti budou do výpočtu histogramu zahrnuty i oblasti, které nejsou označeny jako cílový objekt. Tyto oblasti značně ovlivní získaný histogram a může docházet ke ztrátě sledovaného objektu. Řešením tohoto nedostatku by byla co největší eliminace pixelů, které neoznačují objekt. Aplikováním kernel funkce lze tomuto problému předejít. Tyto funkce určují důležitost každého pixelu v okně. Pixely ve středu okna nabývají největších hodnot, a tedy mají největší vliv na výsledném histogramu. Okrajové pixely, které nejspíš označují oblasti pozadí, naopak nabývají hodnot co nejmenších a tím mají nejmenší vliv na histogram.

Příkladem kernel funkcí jsou Epanechnikov kernel

$$f(n) = \begin{cases} \frac{3}{4}(1 - x^2) & \text{if } |x| < 0 \\ 0 & \text{otherwise} \end{cases} \tag{43}$$

nebo Gaussian kernel

$$f(n) = e^{-\frac{x^2}{2\sigma^2}} \tag{44}$$

Další nevýhodou algoritmu Meanshift je statická velikost sledovací oblasti. Tento problém může mít za následek ztrátu objektu, pokud sledovaný objekt bude vzdálen. Výsledný histogram pak bude z větší části obsahovat oblasti pozadí a bude značně tímto prostředím porušen. Velikost sledovací lze ovlivnit pomocí Bhattacharyya koeficientu(34). Koeficient je vypočítán z cílového a kandidátního histogramu a lze pomocí něj určit změnu velikosti sledovací oblasti. Tento postup však nebyl moc úspěšný, proto ho zde nebudu uvádět.

4.5 Camshift

Algorimus Camshift [18] je odvozen od dříve vynalezeného Meanshift algoritmu. Tento algoritmus byl zaveden roku 1998 autorem Gary Bradski a spadá stejně jako Meanshift do tzv. *jádrových* algoritmů. Jednu z hlavních nevýhod uvedenou v předchozí kapitole pro algoritmus Meanshift byla statická velikost sledovací oblasti tzn. velikost okna nemohla být přizpůsobována velikosti sledovaného cíle. Algoritmus Camshift se snaží tento nedostatek vyřešit. Hlavním rozdílem v algoritmu oproti algoritmu Meanshift je výpočet tzv. zpětné projekce. Výsledkem zpětné projekce je pravděpodobností obraz stejné velikosti

Obrázek 22: Ukázka zpětné projekce

jako vstupní obraz. Tento obraz jednoduše řečeno zobrazuje pixely s větší hodnotou, pokud barva pixelů je obsažena ve sledovací oblasti objektu. Pro výpočet pravděpodobnostního obrazu je nutné stejně jako u Meanshift vypočítat histogram počáteční sledované oblasti. Tento histogram bude uchován po celou dobu sledování objektu a bude sloužit k výpočtu zpětné projekce v každém příchozím obraze. Stejně jako u metody MeanShift se zde využívá výpočet histogramu pro barevný obrázek. Dále je nutné pro výpočet pravděpodobnostního obrazu vypočítat histogram celého vstupního obrazu. Tento krok je oproti Meanshiftu rozdílný a časově náročnější (u Meanshiftu stačilo vypočítat nový histogram jen pro sledovanou oblast). Tyto dva vypočtené histogramy se budou podílet na výsledné zpětné projekci. Pro každou souřadnici bodu vstupního obrazu bude vypočítána pozice v histogramu. Podílem četností výskytu daných pixelů v jednotlivých histogramech zjistíme výslednou hodnotu, která bude zapsána na stejnou pozici do prav-

děpodobnostního obrazu. Tento krok se bude provádět pro každou souřadnici vstupního obrazu. Na výsledný pravděpodobnostní obrázek je možné aplikovat ještě operaci normalizace pro roztažení hodnot do intervalu 0-255. Vypočtená zpětná projekce se bude podílet na výpočtech jednotlivých momentů.

Momenty u algoritmu Meanshift byly vypočetny dle vztahu v kapitole 4.3, kde za funkci $I(x, y)$ byla dosazována hodnota odmocniny podílu p a q histogramu. U algoritmu Camshift funkci $I(x, y)$ nahradí hodnota získaná z pravděpodobnostního obrazu. Po konvergenci sledovací oblasti je nutné provést adaptaci sledovací oblasti na velikost objektu. Pro tuto adaptaci je nutno provést výpočet momentů M_{20} (40), M_{02} (41) a M_{11} (39). Velikost jednotlivých os objektů vypočítáme dle vztahu

$$L = \sqrt{\frac{(A + C + \sqrt{B^2 + (A - C)^2})}{2}}, \tag{45}$$

$$S = \sqrt{\frac{(A + C - \sqrt{B^2 + (A - C)^2})}{2}}, \tag{46}$$

kde $A = \frac{M_{20}}{M_{00}} - x_c^2$, $B = \frac{M_{11}}{M_{00}} - x_c y_c$, $C = \frac{M_{02}}{M_{00}} - y_c^2$. Pro výpočet natočení osy L použijeme vztah

$$\theta = \frac{1}{2} \arctan\left(\frac{2B}{A - C}\right) \tag{47}$$

5 SIMD

Součástí této práce je také paralelizace pomocí vektorových instrukcí. Instrukce jsou využívány pro urychlení některých částí kódu obsažených v metodě Hog a SVM. Samostatné částí kódu, které byly urychleny spolu s grafy, jsou prezentovány v kapitole testy a experimenty. Tato kapitola se skládá z několika podkapitol. Nejprve bude uvedena podkapitola, která se zabývá základním rozdělením paralelních systémů podle Flynnovy klasifikace. Další část se bude věnovat vývoji od instrukcí MMX až po nejnovější AVX. V této části bude také zobrazen kód s instrukcemi (assembler), které vygeneruje kompilátor v režimu release a debug.

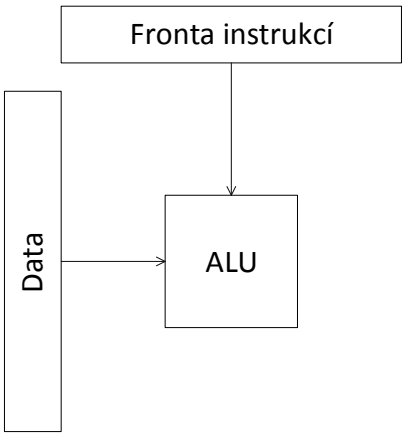
5.1 Flynnova klasifikace

Vstupem mnoha algoritmů můžou být velice rozsáhlá data, nad kterými je nutné provést mnohdy velice náročný výpočet. Většina aplikací je sestavena pro běh v real-time prostředí, kde je kladen důraz na sestavení efektivního kódu, který bude umět počítat s takto velkými daty. Ve většině případu jsou algoritmy urychlovány pomocí paralelizace, kde jsou využívány všechny jádra (procesory), kterým počítač disponuje. Takto sestavený algoritmus dokáže přinést velké urychlení. Před samotnou paralelizací je nutné ještě algoritmus optimalizovat tak, aby podával největší výkon v rámci jednoho jádra. Taková optimalizace může být provedena využitím vektorového procesoru nebo vektorového počítače využitím vektorových instrukčních sad.

Podle Flynnovy klasifikace [19] lze paralelní systémy rozdělit do několika skupin.

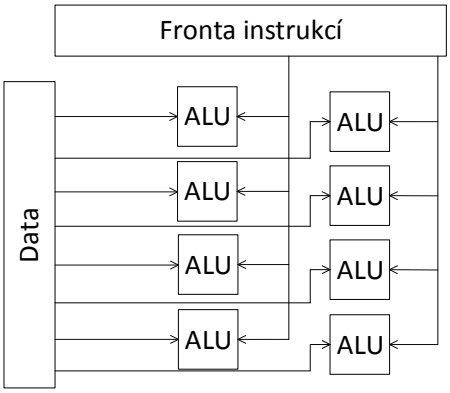
1. SISD – označuje skupinu, kdy data jsou zpracovávána sériově podle jednoho programu.
2. MIMD – jedná se o víceprocesorový systém, kde každý procesor zpracovává jiný program nad různými daty.
3. SPMD – všechny procesory vykonávají stejný program, ale jsou navzájem nezávislé.
4. SIMD – všechny procesory pracují nad jedním programem se stejnou instrukcí, ale nad různými daty.
5. MSIMD – jedná se o několik nezávislých SIMD. Jednotlivé podsystémy zpracovávají svůj program.

Kategorie *SISD* je označována za nejjednodušší implementaci procesorů z technického hlediska, tak i jednoduchými nároky na překladač a programátora. Do této kategorie můžeme zařadit mikroprocesory s architekturou CISC a RISC. Princip je založen na sekvenčním načítání instrukcí z paměti a sekvenčním provádění daných instrukcí. Díky tomu je tato architektura z hlediska rychlosti nedostatečná. Pro zvýšení rychlosti se zde využívá instrukční pipeline, pro dosažení zpracování instrukce alespoň za jeden takt.



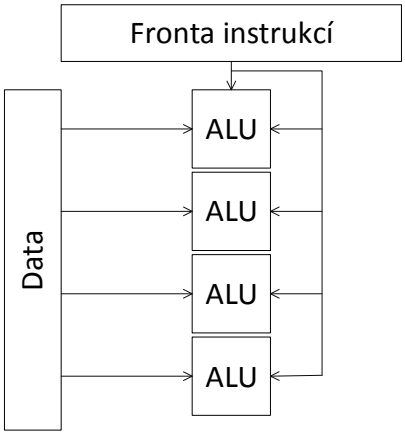
Obrázek 23: Princip SISD instrukce

Kategorie *MIMD* k dosažení paralelismu využívá více procesorů narozdíl od kategorie SISD. Každý procesor pak může vykonávat různé části kódu s různými daty nezávisle na sobě. Tato kategorie je dále rozlišována podle typu paměti na sdílenou a distribuovanou. Sdílený paměťový model využívá pro všechny procesory globální paměť, ke kterou mají přístup všechny procesory. To může mít za následek degradaci výkonu při přístupu procesoru na stejné paměťové místo. Narozdíl od sdíleného paměťového modelu u distribuovaného paměťového modelu každý procesor využívá ke své činnosti svou soukromou paměť a nelze tedy přistupovat přímo k paměti jiného procesoru. Data paměti daného procesoru jsou dostupné pro jiné procesory pouze pomocí komunikace mezi procesory pomocí tzv. zpráv. Nevýhoda u této implementace je časové náročnější přenos dat z jednoho procesoru na druhý. Z pohledu programátora je sdílený paměťový model jednodušší k implementaci programu.



Obrázek 24: Princip MIMD instrukce

Do kategorie *SIMD* instrukcí lze zařadit takové architektury, které pomocí jedné instrukce dokáže zpracovat najednou více dat. Narozdíl od kategorie MIMD zde nedochází ke skutečnému paralelismu (více procesů zpracovaných najednou), ale pouze datovému paralelismu (pouze jeden proces v daný čas). Pro příklad instrukční sada MMX dokáže pomocí jedné instrukce sečíst vektor o velikosti osmi osmibitových nebo čtyř šestnáctibitových hodnot. Taková paralelizace může přinést značné urychlení kódu. Například při práci s barevným obrázkem lze načíst a zpracovat všechny kanály pixelu najednou. Zároveň však přináší mnoho nevýhod, jak z pohledu programátora, kde je nutné přizpůsobit algoritmus pro využití vektorových operací, tak z technického hlediska.



Obrázek 25: Princip SIMD instrukce

Tato klasifikace přináší mnoho nedostatků. Kromě zvýšené spotřebě elektrické energie kvůli implementaci nových registrů, tak i při programování těchto instrukcí je nutné dávat pozor na mnoho věcí, které můžou být velmi omezující.

- Data musí být v poli podle využívané instrukční sady správně zarovnaná.
- Pro využívání instrukcí je nutný přesun do vektorových registrů. Tato operace může být v některých případech velmi neefektivní.
- Instrukční sady neobsahují všechny instrukce potřebné pro běh algoritmu.

5.2 Vývoj SIMD

Jedním z prvních superpočítačů s vektorovým procesorem byl CDC Star 100 představen roku 1964. Superpočítač CDC Cyber 205, který vycházel z CDC Star 100, byl typu paměť-paměť (tento typ vektorového procesoru neobsahoval vektorové registry, proto načítání a následné ukládání bylo realizováno z paměti do paměti, což mělo za následek menší zpoždění pro vykonání operace) a jeho první verze byla zkonstruována roku 1981. Další z úspěšných řad vektorových superpočítačů byly počítače CRAY. První počítač byl uveden

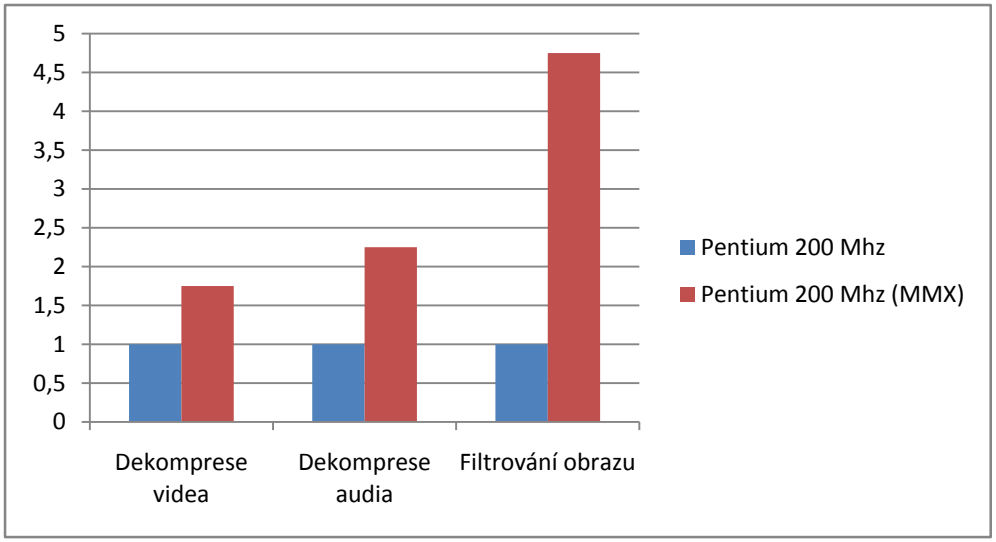
roku 1976 a nesl název CRAY – 1. Tento počítač stejně jako CDC Cyber 205 podporoval operace typu scatter-gather (tyto operace dovolovaly vybrat z vektoru jen část prvků, se kterými byla následně provedena operace) a jako první podporoval řetězení instrukcí. V dalších letech docházelo k vydávání nových verzí počítačů CRAY, u kterých docházelo k vylepšování a zrychlování operací.

První instrukční sada pro desktopové počítače se nazývala MMX a byla vydána roku 1996 firmou Intel. Tato sada byla určena pouze pro procesory typu x86 a první procesor, který měl tuto sadu implementovanou, měl název Pentium P55C. Instrukce v této sadě dokázaly provádět aritmetické a bitové operace s celočíselnými operandy. Dokázala však také zvládat operace s pevnou řadovou čárkou. Největším problémem s návrhem této instrukční sady byl s registry a tím nutností zvýšit počet registrů a plochu čipu. Tento problém byl vyřešen využitím registru, které byly využívány matematickým koprocesorem.

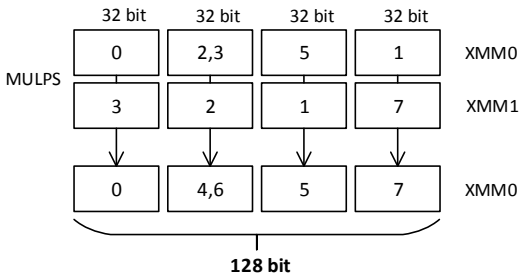
Na instrukční sadu MMX začala reagovat konkurenční firma AMD a vytvořila novou instrukční sadu 3DNow. Tato sada vychází ze instrukční sady MMX a hlavní předností je možnost pracovat s plovoucí řadovou čárkou i s využitím původních MMX registrů. Toho bylo možné dosáhnout uložením 32bitových čísel s plovoucí řadovou čárkou do 64 bitového registru MMX. Kromě nových datových typů přibyly také nové instrukce pracující jak s celočíselnými hodnotami tak s plovoucí řadovou čárkou [20].

Instrukční sady MMX a 3DNow byly vyráběny vždy společností jedné firmy. Příchod nové instrukční sady SSE znamenal spolupráci dvou firem Intel a AMD. Největší novinkou této sady je nová sada registrů, u kterých byla zvětšena kapacita z 64 bitů (MMX, 3DNow) na 128 bitů. Přibyly také nové instrukce zejména pro práci s novými registry. Dosud bylo možné pracovat s instrukcemi, které podporovaly jen cela čísla nebo plovoucí řadovou čárkou s jednoduchou přesností. Přelom přišel s instrukční sadou SSE2, která dovolovala pracovat s plovoucí řadovou čárkou s dvojitou přesností (double). To sebou přineslo vznik nových instrukcí, využívající právě čísla typu double a instrukce, které umožňovala konverzi tohoto datového typu. V roce 2004 byly na trh uvedeny nové procesory s podporou instrukční sady SSE3. Největší novinkou této sady je možnost pracovat s registry horizontálním způsobem a umožnila tak sečíst hodnoty v rámci jednoho registru. Nejnovější instrukční sada, která byla poprvé plně podporována na procesorech Sandy Bridge, nese název AVX. Hlavní změnou oproti předchozím sadám je nová sada registrů, která je zvětšena na dvojnásobnou velikost proti SSE na 256 bitů. Nově byla také přidána možnost třetího operandu u instrukcí a tím je možné uložit výsledek operace do nového registru. U starších instrukčních sad bylo realizováno například sčítání dvou čísel způsobem $a = a + b$ u AVX je možné provést sčítání s uložením do nového registru způsobem $c = a + b$. Rozšíření registrů přineslo také vznik nových instrukcí pracujících s 256 bity.

Využívání těchto instrukční sad v programu může v závislosti na použité instrukční sadě některé algoritmy zásadně urychlit. Správné navržení algoritmu může umožnit kompilátoru překlad z vyššího programovacího jazyku do jazyka strojového s využitím instrukcí SIMD bez nutnosti zásahu programátora. Nejvýznamnější uplatnění těchto instrukcí se našlo v audio a video kodecích, v grafice (raytracing) nebo ve zpracování



Obrázek 26: Ukázka zrychlení pomocí MMX instrukcí na různých příkladech



Obrázek 27: Princip MULPS instrukce

obrazu. V kapitole (7.3) jsou zobrazeny experimenty s instrukční sadou SSE a AVX. Ve výpisu zdrojového kódu je vidět výstup kompilátoru po zkompilování skalárního součinu v debug a release režimu. Jak lze z výpisu vidět, release režim poskytnul po zkompilování dle očekávání mnohem lepší výsledky, proto výsledné testy budou probíhat právě v tomto režimu. Výpis zdrojového kódu v jazyce assembler také poskytuje možnost vidět

využití SSE instrukcí. Instrukce MOVAPS slouží pro přesun 128 - bitů z hlavní paměti do speciálního 128 - bitového registru XMM0. Instrukce MULPS a ADDPS slouží pro vynásobení respektive sečtení dvou 128 - bitových registrů.

mov	eax,dword ptr [k]	movaps	xmm1,xmmword ptr [edx+eax]
mov	ecx,dword ptr [ebp−0ACh]	mulps	xmm1,xmmword ptr [eax]
movaps	xmm0,xmmword ptr [ecx+eax∗4]	add	eax,10h
movaps	xmmword ptr [ebp−210h],xmm0	addps	xmm2,xmm1
movaps	xmm0,xmmword ptr [ebp−210h]	dec	ecx
movaps	xmmword ptr [X],xmm0		
mov	eax,dword ptr [k]		
mov	ecx,dword ptr [features]		
movaps	xmm0,xmmword ptr [ecx+eax∗4]		
movaps	xmmword ptr [ebp−1F0h],xmm0		
movaps	xmm0,xmmword ptr [ebp−1F0h]		
movaps	xmmword ptr [Y],xmm0		
movaps	xmm0,xmmword ptr [X]		
mulps	xmm0,xmmword ptr [Y]		
movaps	xmmword ptr [ebp−1D0h],xmm0		
movaps	xmm0,xmmword ptr [ebp−1D0h]		
movaps	xmmword ptr [X],xmm0		
movaps	xmm0,xmmword ptr [X]		
addps	xmm0,xmmword ptr [Z]		
movaps	xmmword ptr [ebp−1B0h],xmm0		
movaps	xmm0,xmmword ptr [ebp−1B0h]		
movaps	xmmword ptr [Z],xmm0		

Výpis 1: Část výpisu skalárního součinu. Na levé půlce jsou instrukce vygenerované v debug režimu na pravé v release režimu

6 Implementace

Hlavním cílem této práce byla implementace metody Hog pro detekci rukou. V této kapitole se pokusím shrnout některé důležité body implementace, které vedly k vytvoření a vylepšení celé aplikace. Nad rámec této práce byly v teoretické části představeny také některé metody pro sledování objektů. Základní body implementace dvou vybraných algoritmů (MeanShift a CamShift) budou zde také uvedeny. Z důvodu zjednodušení těchto sledovacích algoritmů jsem se rozhodl tyto algoritmy implementovat v jazyce C#. Metoda Hog však byla podle zadání práce vytvořena v jazyce C++ s využitím vektorových instrukcí. V rámci aplikace byla také využita knihovna OpenCV pro základní operace s manipulací se vstupním obrázkem (načítání, normalizace). Pro operaci s obrázky v jazyce C# byla vybrána knihovna EmguCV, která je pouhým *wrapperem* nad knihovnou OpenCV.

6.1 Implementace HOG

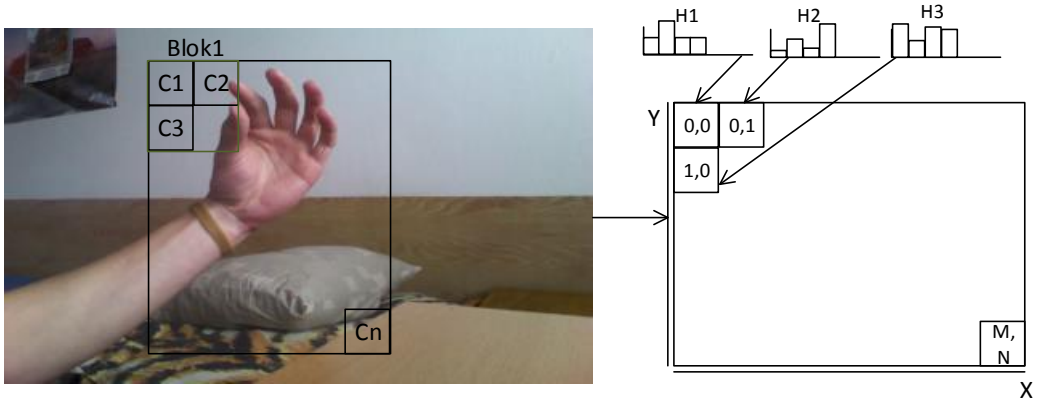
V teoretické části této práce byly představeny některé důležité poznatky, které jsou podstatné pro implementaci celé této práce. Jak už bylo zmíněno v teoretické části, tak základní myšlenou celého algoritmu je popsat obrázek pomocí gradientů hran. K tomuto popisu jsou důležité dva vztahy pro směr gradientu (10) a pro velikost gradientu (11). Výsledky těchto vztahů budou sloužit pro sestavení celého histogramu. Celý popis jak probíhá sestavení výsledného histogramu je uveden v podkapitole *(Popis metody Hog)*.

Pro orientaci zde uvedu jenom základní kroky implementace algoritmu Hog:

1. Předzpracování obrazu
2. Výpočet gradientů jednotlivých pixelů
3. Výpočet úhlu a velikosti gradientu
4. Umístění jednotlivých gradientu do histogramu jednotlivé mřížky podle velikosti úhlu a inkrementace dané pozice histogramu o velikost gradientu
5. Normalizaci skupiny mřížek
6. Seskupení všech histogramů do jednoho a následná klasifikace

Celý algoritmus je poměrně přímočarý a nelze v něm najít mnoho způsobu jak jej urychlit. První krok celého algoritmu je prováděn pomocí knihovny OpenCV. Obrázek je načten pomocí funkce *cvLoadImage*. Poté je převeden do jedno-kanálového režimu a následně je normalizován do rozsahu od 0-255 pomocí funkce *cvNormalize*. Následující krok výpočtů gradientů je tedy prováděn na černobílém a normalizovaném obrázku. Výpočet gradientu je prováděn pomocí filtru (vztah 12). Výpočet podle zmíněného vztahu je jednoduchý a přináší dobré výsledky. V aplikacích pro detekci hran je využíván nejčastěji výpočet gradientu pomocí Sobelova operátoru (8), ale z důvodu náročnějšího výpočtu a stejné kvality detekce bylo lepší zvolit méně náročný výpočet. Provedení kroků 2, 3 a 4 nabízí poměrně lehký způsob výrazného zrychlení výpočtu příznaků. Jak už bylo zmíněno v

teoretické části pro detekci rukou v obraze je nutné prozkoumat poměrně velké množství oken v obraze, která se navzájem překrývají. Dále bylo také zmíněno, že bloky se můžou překrývat, což znamená, že jedná mřížka je použita v několika blocích. Špatný návrh algoritmu by mohl znamenat poměrně časté počítání histogramu stejné mřížky v několika různých oknech nebo blocích. Z tohoto důvodu je nutné předzpracovat si výpočet histogramů v jednotlivých mřížkách a v různých detekčních oknech a blocích využívat právě tyto předzpracované data. K uložení těchto informací je v programu vytvořeno trojrozměrné pole, kde osy x a y označují pozici mřížky. Histogram, který je vytvořen z dat získaných ze vztahů (10) (11), je uložen na ose z příslušné x a y souřadnice. Pro lepší představu je princip předzpracování zobrazen na obrázku (28).



Obrázek 28: Princip uložení informací do třírozměrného pole

Kromě již zmíněného urychlení byla tato část převedena také do vektorových instrukcí. V této části jsem se při implementaci vektorových instrukcí potýkal pouze s jednou malou z mnoha nepříjemnostmi, které byly uvedeny v kapitole *SIMD*. Pro výpočet směru gradientů je potřeba využívat instrukci *atan2*, která ovšem není implementována v instrukční sadě SSE ani AVX. Z tohoto důvodu bylo nutné vytvořit vlastní implementaci této instrukce v instrukční sadě SSE. Algoritmus pro výpočet funkce *atan2* byl přezvat ze zdroje [21]. Hlavním krokem v převodu z klasických do vektorových instrukcí funkce *atan2* byla nutnost odstranit rozhodovací instrukce *if* a *else*, které by v převodu do vektorových instrukcí znamenaly velký problém. Poté už stačilo pouze nahradit klasické instrukce vektorovými instrukcemi (funkcemi).

Dalším krokem v algoritmu Hog je nutnost normalizace jednotlivých mřížek. V této práci jsou implementované dvě normalizační techniky L1-norm(14) a L2-hys(16). V kapitole *Testy a experimenty* byla využita k testování normalizace pouze L2-norm, protože podávala lepší výsledky podle [10] a vlastního testování. Tato normalizace byla také pomocí vektorových instrukcí urychlena a otestována spolu se získávaní příznaků v kapitole *Testy a experimenty*. V převodu do vektorových instrukcí nastal největší problém při velikosti pole, které nebylo dělitelné čtyřmi nebo osmi. Z toho důvodu musela být

poslední část daného pole vypočítána pomocí klasických instrukcí, což způsobilo vetší režii a tudíž větší časovou náročnost. Posledním krokem celého algoritmu je sesbírání všech normalizovaných histogramů do jednoho a jeho následná klasifikace pomocí SVM. Klasifikace pomocí SVM byla v rámci této práce také implementována a urychlená pomocí instrukcí SSE a AVX. V podkapitole SVM byly popsány různé vztahy pro nalezení lineárního oddělovače. V rámci této práce jsem zkoušel dvě jádrové funkce lineární (28) a RBF (30), ale v testování byla využita pouze funkce lineární, která podávala podobné výsledky jako funkce RBF.

6.2 Implementace MeanShift a CamShift

Teoretická část zabývající se sledováním objektů popisuje několik metod vhodných ke sledování ruky. V této části byly zmíněné také důvody proč je vhodné ke sledování ruky využít metody, které nezkoumají tvar objektů, ale zabývající se barvou daného objektu. Dále jsou v teoretické části uvedeny teoretické poznatky, které jsou nezbytné pro implementaci. Nad rámec této práce jsem tedy implementoval algoritmy MeanShift a Camshift, které jsou následně otestovány v kapitole *Testy a experimenty*. Algoritmus těchto metod je poměrně přímočarý a nenašel jsem v něm místa, která by stejně jak u metody Hog nabízela možnosti urychlení. Proto další text bude popisovat důležité části programu ve spojitosti s teorií.

Základem celého algoritmu Meanshift je třída *MeanShift*, kde její metoda *Track* (pseudokód (2)) je hlavní metoda této třídy a provádí konvergenci daného sledovacího okna. Dále program obsahuje třídu *Histogram*, která obsahuje veškeré metody pro práci s histogramem. Nejdůležitější součástí třídy *Histogram* jsou metody pro výpočet histogramu. Pro práci s metodou MeanShift jsou důležité dvě metody pro výpočet histogramu *CreateHistogram* a *CreateHistogramKernel*. První zmíněná metoda vypočítává histogram na základě vztahu (33). Druhá metoda obsahuje výpočet, kde jednotlivé pixely jsou ještě ohodnoceny pomocí kernel funkce. Do této metody jsme vybral výpočet pomocí *Epanechnikova* kernelu (43) (pro připomenutí kernel zajišťuje vetší váhu pro pixely uprostřed okna). V rámci testování a porovnávání s metodou Camshift jsem zvolil výpočet histogramu právě pomocí metody *CreateHistogramKernel*, který by z důvodů zmíněných v teoretické části měl poskytovat lepší výsledky. Nezbytnou třídou pro algoritmus Meanshift je třída *Moment* a její metoda *GetMoments*, která přijímá dva histogramy (histogram vytvořený na základě nalezeného objektu a histogram nově posunutého okna) a vrací strukturu *MomentData*, která obsahuje vypočítané jednotlivé momenty.

```
Vstup: Cilovy histogram, pozice okna (X,Y,W,H), min_dist, max_iter
Vystup: Nova pozice okna
do
{
   Vypocitej kandidatni histogram
   Vypocitej momenty M10, M01 a M00
   Xc = M10 / M00 a Yc = M01 / M00
   Nastav stred okna na pozici Xc a Yc
   vzdalenost =   vzdalenost_oken(X,Y,Xc,Yc)

}while(vzdalenost > min_dist && iter < max_iter)
```

Výpis 2: Pseudokód algoritmu Meanshift

Metoda *Track* (pseudokód (2)) využívá všechny tyto třídy a jejich funkce pro konvergenci okna do místa sledovaného objektu. Vstupem metody *Track* je odkaz na histogram, který je vypočten pomocí funkce *CreateHistogramKernel*. Hlavním bodem metody *Track* je cyklus, který provádí konvergenci daného okna do místa sledovaného objektu. Na začátku cyklu je proveden výpočet histogramu nově vypočítané pozice okna a na základě vstupního a nově získaného histogramu jsou vypočítány jednotlivé momenty (*M10, M01 a M00*), které určí výsledný střed okna. Tento cyklus je prováděn tak dlouho dokud je vzdálenost oken větší než minimální vzdálenost nebo není překročen počet iterací.

```
Vstup: Pravdepodobnostni obraz, pozice okna (X,Y,W,H), min_dist, max_iter
Vystup: Nova pozice okna
do
{
   Vypocitej kandidatni histogram
   Vypocitej momenty M10, M01 a M00 z pravdepodobnostni obraz
   Xc = M10 / M00 a Yc = M01 / M00
   Nastav stred okna na pozici Xc a Yc
   vzdalenost =   vzdalenost_oken(X,Y,Xc,Yc)

}while(vzdalenost > min_dist && iter < max_iter)
Adaptuj velikost okna
```

Výpis 3: Pseudokód algoritmu Camshift

Hlavní změnou oproti algoritmu Meanshift je v algoritmu CamShift výpočet pravděpodobnostního obrazu, který probíhá ve třídě *CamShift*. Tento výpočet je ze všech části programu nejnáročnější, proto alespoň zde byla zvolena možnost pro jednoduchou paralelizaci. Pro výpočet algoritmu Camshift byla třída *Moment* rozšířená o výpočet momentů pomocí pravděpodobnostního obrazu (kapitola *Camshift*). Narozdíl od algoritmu Meanshift je v metodě *Track* po konvergenci okna do místa sledovaného objektu provedena adaptace velikosti okna. Celý algoritmus je oproti těmto změnám totožný s algoritmem MeanShift.

7 Testy a experimenty

Tato část práce bude věnována výsledkům implementací jednotlivých kapitol popsaných výše. První část je věnována testům metody Hog. Srovnává různé nastavení detekčního okna a porovnává dosažené výsledky vlastní implementace s implementací pomocí OpenCV knihovny. Dále je v této části popsána metodika pro testování jednotlivých metod. Další část této kapitoly je věnována porovnávání implementace metod pro sledování objektů MeanShift a CamShift. Pro porovnání výsledků této části bylo nutné také provést manuální označení středu sledovaného objektu, proto zde můžou vidět malé odchylky, i když sledování probíhalo v pořádku. Výsledky sledování objektů jsou vyneseny do grafu, který zobrazuje informace o skutečné a nalezené pozici objektu. Poslední podkapitolou v této části je porovnání výsledků urychlení pomocí speciálních instrukcí SSE a AVX. Tato část je rozdělena na několik úseků. První úsek zkoumá možnosti urychlení extrakci příznaků. Další úsek zobrazuje výsledky pro urychlení samotné klasifikace pomocí metody SVM. Nakonec je zde zobrazen graf, který zobrazuje urychlení celého algoritmu Hog a SVM.

7.1 Hog

V této podkapitole budou prezentovány výsledky detekce rukou pomocí metody Hog. Jelikož cílem této práce nebylo vytvořit detektor, pracující v komplexním prostředí, ale zaměřit se pouze na konkrétní prostředí, proto trénovací a testovací snímky jsou pořízeny z jednoho prostředí.

Pro trénovaní detektoru byl vytvořen vlastní dataset obrázků, který obsahuje 689 positivních a 2500 negativních obrázků o rozměrech 64x128 a 96x128 pixelů. Positivní obrázky zachycují v popředí ruku v různých polohách. Pozadí těchto obrázků je tvořeno různými částmi daného prostředí. Pro testování byly vytvořeny dvě kolekce obrázku. První kolekce obsahuje objekty (ruce), které se hodně podobají obrázkům vytvořeným v trénovací sadě. Druhá kolekce obsahuje positivní obrázky, které jsou sice ze stejného prostředí, ale podobnost obrázků z trénovací sady je hodně vzdálená. Negativní obrázky této kolekce jsou vytvořeny v úplně odlišném prostředí, než v jakém byly získané trénovací obrázky.

V kapitole popisující metodu Hog byl popsán algoritmus, ve kterém byly popsány možnosti různého nastavení pro získání příznaků. Toto nastavení bude předmětem zkoumání ve výše popsaných kolekcích testovacích obrázků. Kromě vlastní implementace metody Hog bude testována také možnost získávání příznaku ze třídy *HogDescriptor* knihovny OpenCV. U této metody se jak pro trénování, tak i pro testování využívá stejná implementace SVM.

Celková úspěšnost detekce bude počítána dle vzorce

$$A_{cc} = \frac{TP + TN}{TP + TN + FP + FN} \tag{48}$$

TP (true positive) označuje správnou detekci ruky

TN (true negative) označuje správnou detekci pozadí

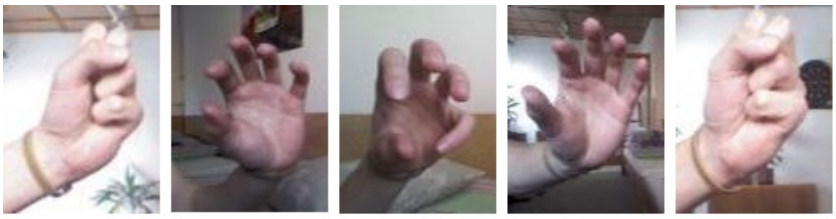
FP (false positive)označuje nedetekovanou ruku

FN (false negative) označuje nedetekované pozadí

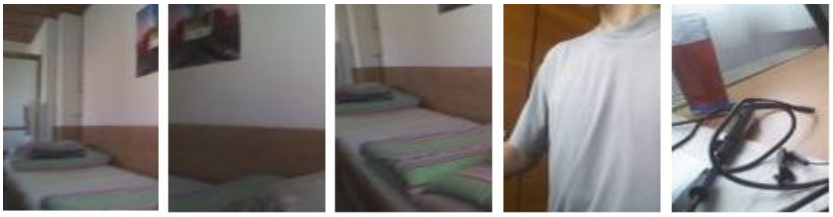
Tabulka (1) obsahuje různé možnosti nastavení detekčního okna. Toto nastavení je zaměřené především na změnu velikosti mřížek, normalizovaných bloků a počtu binů, které ze všech možných parametrů přinášelo největší změny detekce rukou. Pro testování byla vybraná možnost normalizace bloků pomocí L2-norm (vztah 16), která poskytovala nejlepší výsledky ze všech uvedených v kapitole Hog jak podle vlastního testování, tak podle [10]. Trénování a klasifikace byla testována pomocí SVM s lineárním kernelem. Nastavování různých kernelů uvedených v podkapitole SVM nepřinášelo žádné výraznější zlepšení či zhoršení klasifikace.

Tabulka 1: Nastavení parametrů detekčního okna

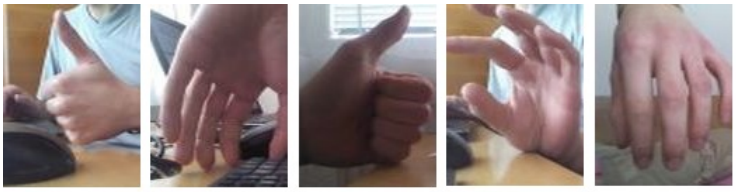
	Nastavení parametrů			
	Velikost SW	**Velikost mřížky**	**Velikost bloku**	**Počet binů**
Konfigurace1	64x128	8x8	4x4	12
Konfigurace2	64x128	8x8	2x2	9
Konfigurace3	64x128	8x8	2x2	12
Konfigurace4	64x128	16x16	2x2	12
Konfigurace5	96x128	8x8	4x4	12
Konfigurace6	96x128	8x8	2x2	9
Konfigurace7	96x128	8x8	2x2	12
Konfigurace8	96x128	16x16	2x2	12



Obrázek 29: Ukázky positivní testovací sady první kolekce



Obrázek 30: Ukázky negativní testovací sady první kolekce



Obrázek 31: Ukázky positivní testovací sady druhé kolekce

Obrázek 32: Ukázky negativní testovací sady druhé kolekce

První kolekce (výsledky v tabulkách (2)(3)) pro testovací účely obsahovala 82 positivních a 120 negativních obrázků. Ve vlastní implementaci metody Hog nejlepší výsledky detekce pro první kolekci byly dosahovány pro konfiguraci 2 a konfiguraci 8. Konfigurace 2 byla převzata od autorů Delal a Trigs, kteří toto nastavení detekčního okna zkoušeli pro detekci osob a přinesla jeden z nejlepších výsledků. U tohoto nastavení je však větší počet příznaků. To znamená delší čas výpočtu jak pro získání, tak i pro výslednou klasifikaci příznaků.

V implementaci OpenCV bylo nejlepších výsledků dosaženo pro konfiguraci 2 a 4. Výsledky těchto konfigurací jsou o něco horší než ve vlastní implementaci, ale výhodou je nízký počet negativních obrázků detekovaných jako ruka.

Tabulka 2: Výsledky pro kolekci 1

	Výsledky pro kolekci 1				
	TP	**FN**	**FP**	**TN**	**Acc**
Konfigurace1	45	37	7	113	0.78
Konfigurace2	71	11	3	117	0.93
Konfigurace3	65	17	3	117	0.9
Konfigurace4	69	13	3	117	0.92
Konfigurace5	34	48	12	108	0.7
Konfigurace6	55	27	6	114	0.83
Konfigurace7	56	26	4	116	0.85
Konfigurace8	72	10	3	117	0.935

Tabulka 3: Výsledky pro kolekci 1 (OpenCV)

	Výsledky pro kolekci 1 (OpenCV)				
	TP	**FN**	**FP**	**TN**	**Acc**
Konfigurace1	32	50	5	115	0.72
Konfigurace2	42	40	0	120	0.8
Konfigurace3	27	55	1	119	0.72
Konfigurace4	47	35	2	118	0.81
Konfigurace5	11	71	2	118	0.63
Konfigurace6	16	76	1	119	0.66
Konfigurace7	11	71	1	119	0.64
Konfigurace8	15	77	2	118	0.65

Druhá kolekce (výsledky v tabulkách (4)(5)) pro testovací účely obsahovala 65 positivních a 117 negativních obrázků. Tato kolekce na rozdíl od první obsahuje negativní obrázky pořízené z jiného prostředí než trénovací sada. Positivní obrázky jsou pořízené sice ze stejného prostředí, ale pozadí na kterém je ruka vyfocena, není tak často obsaženo v trénovací množině.

Na rozdíl od první kolekce byla již z výše zmíněného důvodu mnohem menší úspěšnost klasifikace rukou a také větší počet negativních obrázku bylo označeno jako ruka. Nejlepšího výsledku u vlastní implementace dosahovala konfigurace 3. Tato konfigurace se liší od konfigurace 2, která byla nejvhodnější u první kolekce, větším počtem binů a tím pádem časově náročnějším výpočtem.

OpenCV implementace přináší ve většině případů horší detekce v jednotlivých konfiguracích. Výsledky jednotlivých nastavení jsou si hodně podobny a nelze určit jednoznačného vítěze. Konfigurace 6 však přinesla nejlepší poměr mezi úspěšnou detekcí rukou a detekcí pozadí.

Tabulka 4: Výsledky pro kolekci 2

	Výsledky pro kolekci 2				
	TP	**FN**	**FP**	**TN**	**Acc**
Konfigurace1	21	44	22	95	0.64
Konfigurace2	16	49	11	106	0.67
Konfigurace3	22	43	10	107	0.712
Konfigurace4	23	42	29	88	0.61
Konfigurace5	25	40	29	88	0.62
Konfigurace6	13	52	6	111	0.68
Konfigurace7	16	49	6	111	0.7
Konfigurace8	25	40	27	90	0.63

Tabulka 5: Výsledky pro kolekci 2 (OpenCV)

	Výsledky pro kolekci 2 (OpenCV)				
	TP	**FN**	**FP**	**TN**	**Acc**
Konfigurace1	24	41	30	87	0.61
Konfigurace2	21	44	22	95	0.64
Konfigurace3	20	45	21	96	0.64
Konfigurace4	23	42	23	94	0.64
Konfigurace5	22	43	23	94	0.64
Konfigurace6	23	42	19	98	0.66
Konfigurace7	13	52	12	105	0.65
Konfigurace8	20	45	34	83	0.56

7.1.1 Shrnutí výsledků

Výhodou metody Hog je využití pro detekování mnoha různých objektů bez výrazných zásahů do algoritmu. Nejčastěji detekovanými objekty u této metody jsou automobily nebo osoby, u kterých se dosahuje nejlepších výsledků detekce. Tato metoda v sobě obsahuje také plno nedostatků, které lze určitými kroky vyřešit. Jako hlavní nevýhodu bych uvedl nutnost počítat všechny příznaky v rámci okna, pokud se objekt v daném místě nenachází. Tento problém by se dal vyřešit podobně jako v metodě Viola Jones využitím tzv. kaskadové klasifikaci, kde by se počítala pouze část z uvedených příznaků a k zastavení klasifikace by došlo na začátku celé kaskády bez nutnosti počítat všechny příznaky. Z výsledků v tabulkách uvedených výše lze vidět, že univerzální nastavení pro detekci rukou nelze zvolit. Úspěšnost detekce je dána mnoha faktory od algoritmů, přes různé nastavení získávání příznaku, až po získání správných trénovacích dat. Všechny tyto faktory výrazně ovlivňují výslednou detekci. Detekce rukou je ještě ovlivněna dalším faktorem, který u detekce osob není až tak výrazný a to vysoká variabilita poloh nebo natočení rukou. Tato různorodost může způsobit problémy v detekci, kde ruka může být

označena jako pozadí a naopak. Z výše uvedených testů lze označit jako nejvhodnější konfigurace pro detekce rukou konfigurace 2 a 3. Konfigurace 2 byla také označena jako jedna z nejlepších u detekce osob [10]. Obě konfigurace se liší pouze ve velikosti počtu binů a tím také zvýšeným počtem příznaků u konfigurace 3, která má za následek pomalejší detekci. V rámci velikosti detekčních oken mají konfigurace 2 a 3 největší dimensi vektoru příznaků, proto lze tedy říci, že větší počet příznaků přináší lepší detekce, ovšem za cenu delšího výpočtu jak pro extrakci příznaků, tak i pro klasifikaci.

7.2 Meanshift a Camshift

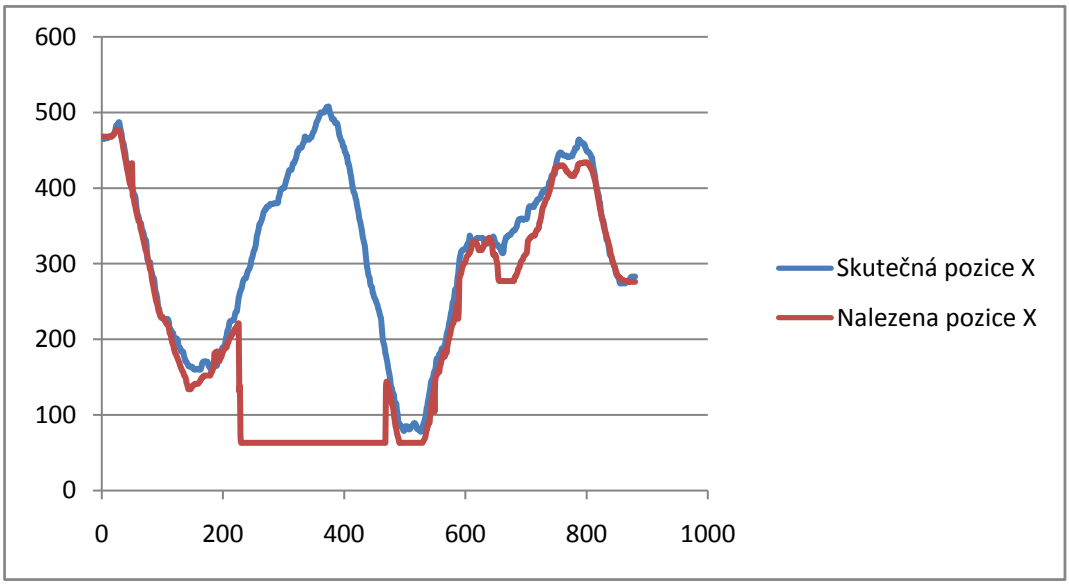
Pro testovací účely byly natočeny dvě videa, kde každé bylo natočeno v rozdílném prostředí. Videa byla natočena v rozlišení 1280 x 720, ale pro sledování bylo nutné kvůli rychlosti tuto velikost snížit na polovinu (640x360). Jednotlivé obrázky videa byly získané pomocí knihovny OpenCV. Pro měření přesnosti sledovaného objektu bylo nutné zjistit skutečnou polohu objektu. Kvůli tomu bylo nutné vytvořit novou jednoduchou aplikaci, ve které byl ručně označen střed sledovaného objektu. Získána data z metody Meanshift a Camshift byla vynesena do grafu, ve kterém je porovnávána se skutečnou polohou objektu. Jednotlivé grafy jsou rozděleny podle metod a podle jednotlivých souřadnic.

První čtyři grafy zobrazují data z prvního videa. Toto video je z pohledu sledovacích algoritmů jednodušší, díky malému překrývání sledovacího objektu s podobně barevným objektem v pozadí. Druhé video zaznamenané v grafech (38) - (41), je kvůli častému překrývání ruky s pozadím podobné barvy a příliš velkému osvětlení složitější. Z obrázku (33) můžeme pozorovat podobnost sledovaného objektu (ruky) s dveřmi a stoly. Tyto předměty se kvůli velké barevné podobnosti s rukou staly jedním z hlavních problému sledování.

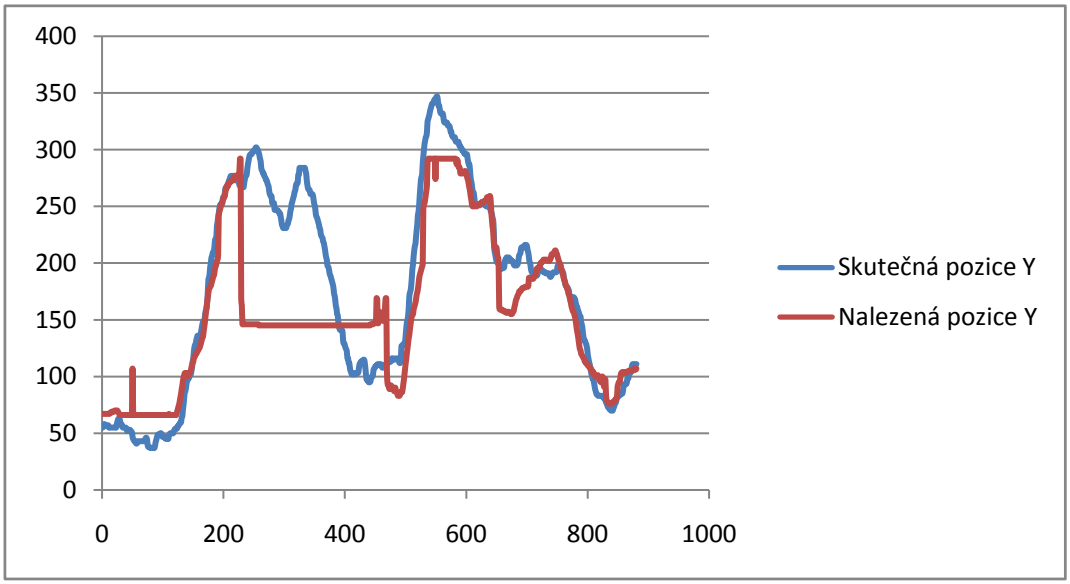
Obrázek 33: Ukázka prostředí ve kterém byly jednotlivé videa natočena, horní obrázek zachycuje první (jednodušší) video, dolní zachycuje ukázku druhého (složitějšího) videa

Grafy (34) a (35) zobrazují průběh pro souřadnici X a Y metody Meanshift. U obou souřadnic byl průběh do 200 snímku téměř totožný. Největší odchylka byla zaznamenána na snímcích 200 – 500, kde došlo ke shodě histogramu ruky s pozadím. Průběh grafu od 500 snímku byl téměř totožný s ideálním průběhem sledovacího algoritmu.

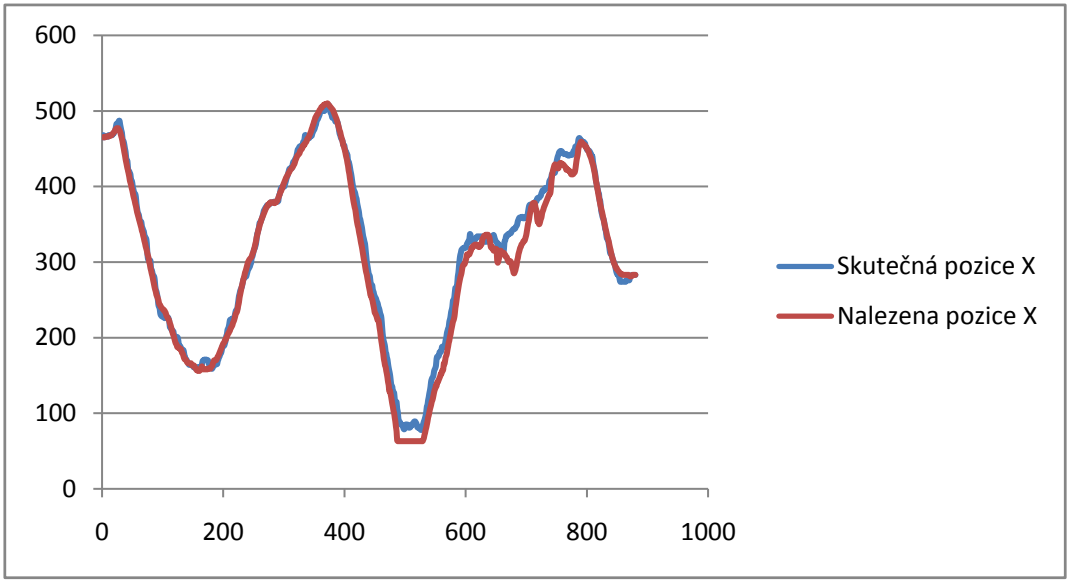
Grafy (36) a (37) představují průběh sledovací oblasti skutečné a nalezené pro metodu Camshift. Tato metoda u tohoto videa projevovala mnohem lepší výsledky než metoda Meanshift. Nedocházelo zde téměř k žádným odchylkám od skutečné polohy objektu. Větší odchylka nastala pouze na souřadnici Y mezi 500 a 600 snímkem, ale ani zde nedošlo ke ztrátě sledovaného objektu.



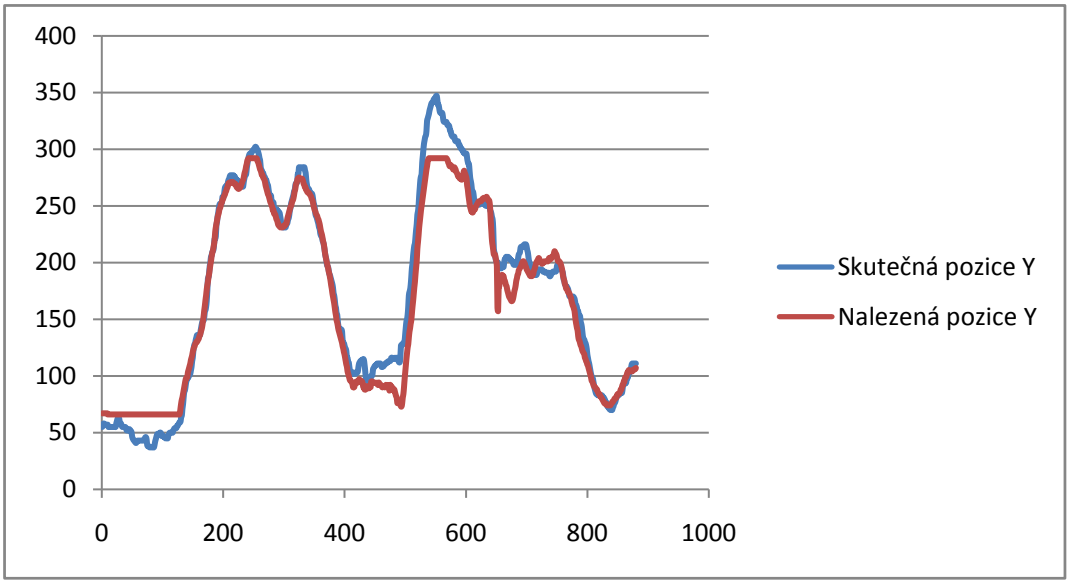
Obrázek 34: Graf nalezené pozice v ose X pro MeanShift



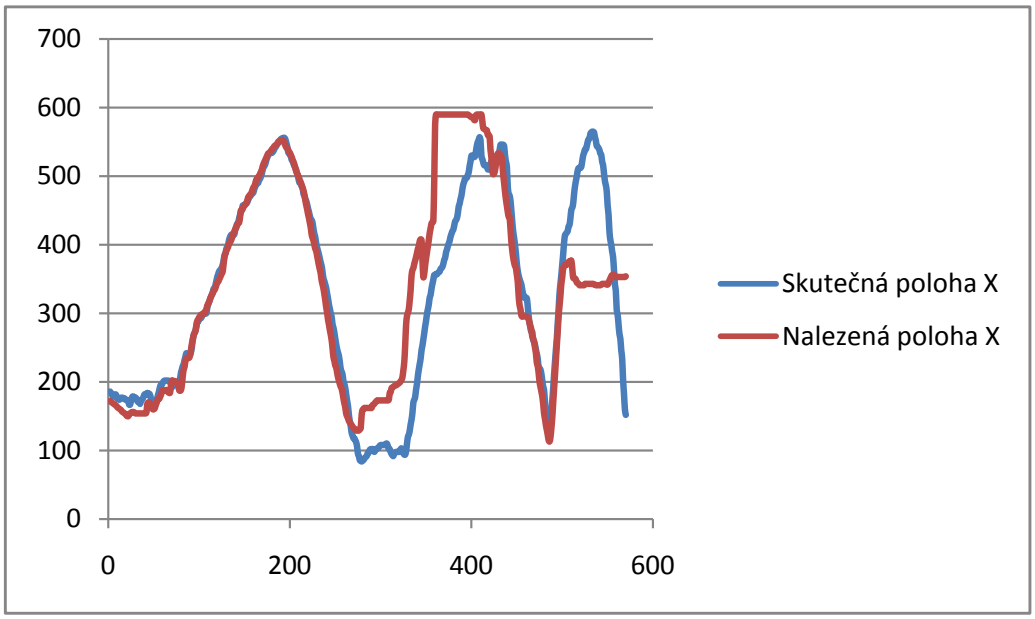
Obrázek 35: Graf nalezené pozice v ose Y pro MeanShift



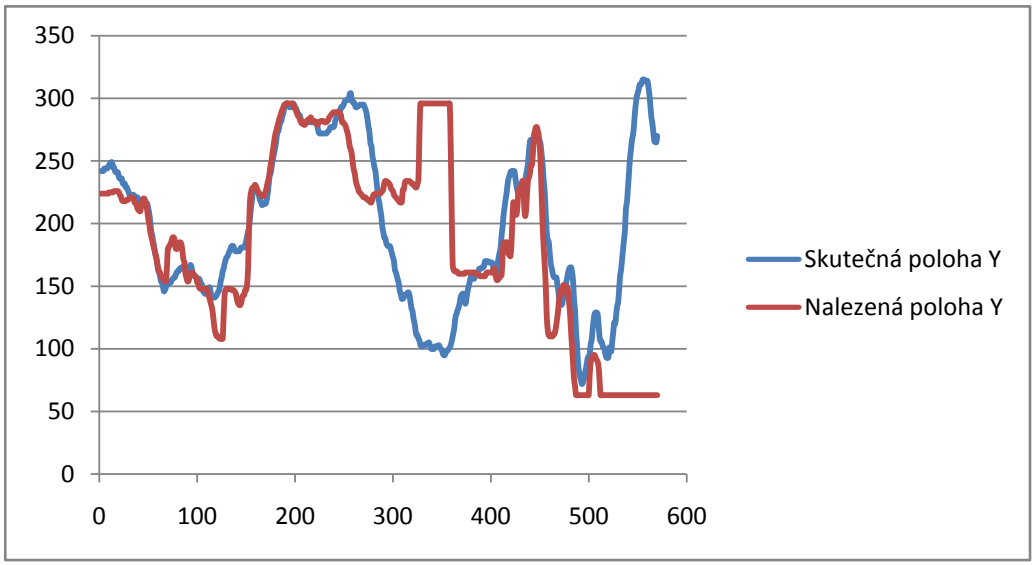
Obrázek 36: Graf nalezené pozice v ose X pro CamShift



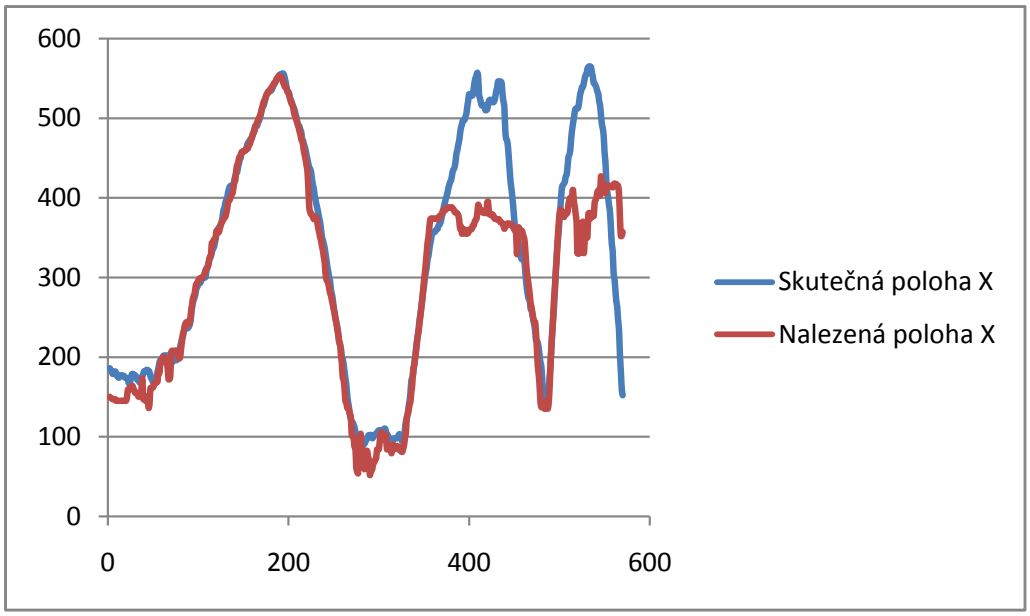
Obrázek 37: Graf nalezené pozice v ose Y pro CamShift



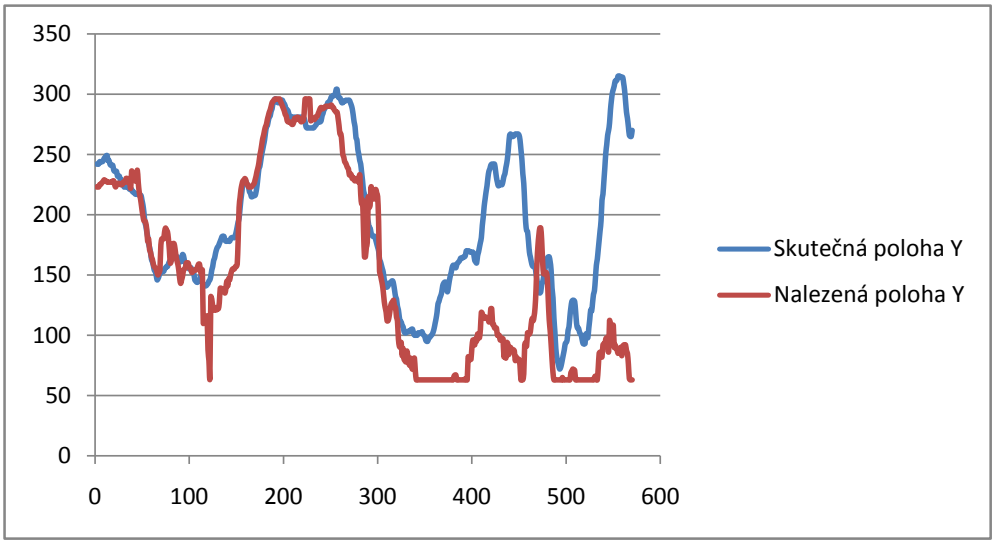
Obrázek 38: Graf nalezené pozice v ose X pro MeanShift



Obrázek 39: Graf nalezené pozice v ose Y pro MeanShift



Obrázek 40: Graf nalezené pozice v ose X pro CamShift



Obrázek 41: Graf nalezené pozice v ose Y pro CamShift

Průběh pro druhé video zachycují grafy (38) - (41). Zde můžeme oproti grafům z prvního videa pozorovat daleko větší odchylky. V metodě Meanshift došlo ke ztrátě sledovaného objektu v ose Y na snímcích 300 - 380 a v obou osách na snímcích 530 až do konce videa. Obě tyto ztráty způsobila konvergence sledovací oblasti ke dveřím. Naštěstí došlo po snímku 380 k znovu nalezení sledovacího objektu.

U metody Camshift došlo také ke dvojí ztrátě sledovacího objektu. První ztráta nastala na snímcích 350 - 450 v obou osách. Druhá ztráta nastala po snímku 500. Stejně jak u metody Meanshift tyto ztráty způsobila konvergence sledovací oblasti ke dveřím.

7.2.1 Shrnutí výsledků

Sledování ruky je velice obtížné a najít proto vhodný algoritmus, který by zvládal sledování v reálném čase, je velmi náročné. Mezi hlavní důvody, kvůli kterým docházelo k častým ztrátám sledovaného objektu (ruky), je nevýrazná barva ruky, kvůli které často docházelo ke konvergenci k jinému objektu než ruka. Jak můžeme pozorovat z výše uvedených textů a jednotlivých grafů, tak lepších výsledků dosahoval algoritmus Camshift především u prvního videa, kde nedošlo k žádné ztrátě sledovaného objektu. U druhého videa nelze označit žádného vítěze. Ke ztrátě sledovaného objektu docházelo na podobných místech. Z výše uvedených textů tedy vyplývá, že algoritmus Camshift poskytoval o něco lepší výsledky. Nutno však podotknou nevýhodu, kterou algoritmus Camshift obsahuje. U jednotlivých videí byly naměřeny také jednotlivé časy výpočtu a počet iterací pro jednotlivé snímky a následně byly tyto údaje zprůměrovány. Výsledky jsou zobrazeny v tabulce (6), kde můžeme odečíst již zmíněnou nevýhodu a to čas potřebný k výpočtu jednoho snímku. Tento čas je asi 4x horší než u algoritmu Meanshift. Tento negativní výsledek je způsoben nutností počítat zpětnou projekci pro každý snímek videa.

Tabulka 6: Srovnání Meanshift a Camshift metody

	Výsledky pro kolekci 2 (OpenCV)	
	Počet iterací	**Výpočet jednoho snímku**
Camshift 1. video	3,55	133 ms
Camshift 2. video	4,36	135 ms
Meanshift 1.video	3,47	30 ms
Meanshift 2.video	4,45	33 ms

7.3 SSE a AVX instrukce

V této podkapitole budou popsány dosažené výsledky urychlení metody Hog pomocí instrukcí SSE a AVX. Testy byly prováděny na notebooku s procesorem Intel Core i5-3230M 2.6GHz. První graf bude zobrazovat výsledky urychlení pro SVM klasifikaci, druhý pro získání příznaků a poslední bude kombinací těchto dvou implementací. Časy naměřené u prvního grafu nejsou reálnými časy běhu v programu. Tyto hodnoty času byly získané simulováním daných části ve smyčce.

V kapitole Experimenty s parametry Hog můžeme pozorovat vysokou dimensi vektoru příznaků, které byly vypočítány z jednotlivých detekčních oken. Podle vztahu (27) pro výsledné klasifikování je nutné provést výpočet skalárního součinu pro různé pozice testovaného obrázku mezi příznaky nalezené v testovacím obrázku a nalezenou nadrovinou. Tato operace z důvodu vysokého počtu příznaků je časově náročná, proto využití instrukcí pro tuto operaci může přinést urychlení algoritmu Hog. Do výsledného porovnání jsem zahrnul také urychlení pomocí techniky "loop unrolling". Cílem této techniky je urychlit vykonávání programu eliminaci podmíněných skoků u smyčky při každé iteraci. Eliminace těchto kontrol je provedena pomocí tzv. rozbalení cyklu. Během jedné iterace se tedy provede více operací.

```
double s = 0;
for( k = 0; k <= sizeFeaturestmp − 4; k += 4 )
   s += sample[k]∗features[k] + sample[k+1]∗features[k+1] +
   sample[k+2]∗features[k+2] + sample[k+3]∗features[k+3];
return s;
```

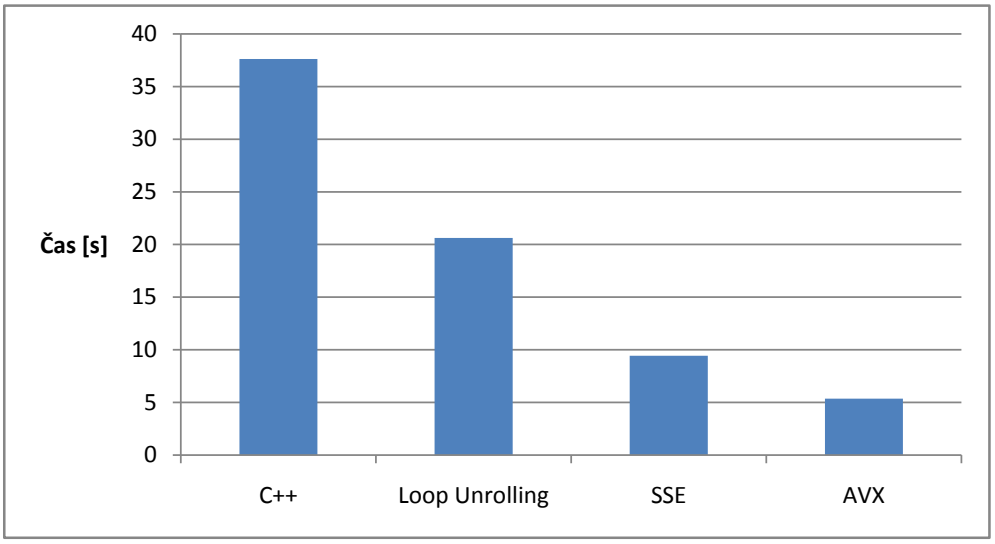
Výpis 4: Loop unrolling pro skalární součin

```
movaps    xmm1,xmmword ptr [ecx+edx]
mulps     xmm1,xmmword ptr [edx]
add       edx,10h
addps     xmm2,xmm1
dec       eax
```

Výpis 5: Kód vygenerovaný kompilátorem pro hlavní smyčku skalárního součinu u instrukcí SSE v release režimu

```
vmovaps   ymm1,ymmword ptr [ecx+edx]
vmulps    ymm1,ymm1,ymmword ptr [edx]
vaddps    ymm0,ymm1,ymm0
lea       edx,[edx+20h]
dec       eax
```

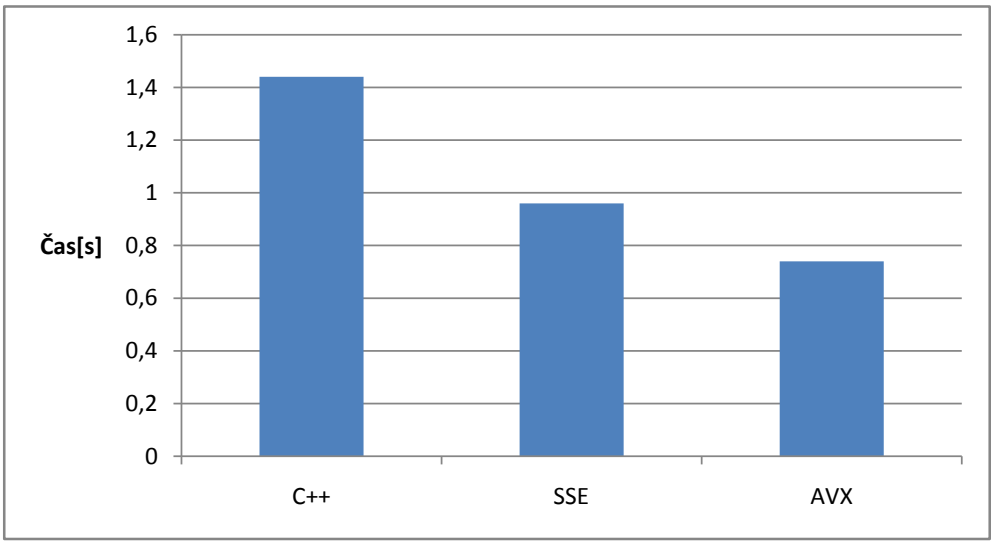
Výpis 6: Kód vygenerovaný kompilátorem pro hlavní smyčku skalárního součinu u instrukcí AVX v release režimu



Obrázek 42: Graf urychlení pomocí SIMD instrukcí a loop unrolling pro operaci skalárního součinu

Z grafu (42) můžeme pozorovat zásadní urychlení pomocí SIMD instrukcí. Důvodem tohoto zrychlení je využití nových registrů o kterých bylo zmíněno v kapitole SIMD. Registry u instrukční sady dokážou pojmout čtyři hodnoty typu float a v rámci jedné instrukce vykonat operace na všech čtyřech hodnotách. Z grafu vyplývá dle očekávání čtyřnásobné zrychlení u instrukcí SSE oproti použití klasických instrukcí. U instrukční sady AVX nabývají registry 256 bitové hodnoty tzn. dokážou uložit osm hodnot typu float. Oproti instrukční sady SSE, která má poloviční velikost registrů, vyplývá z grafu téměř dvojnásobné zrychlení.

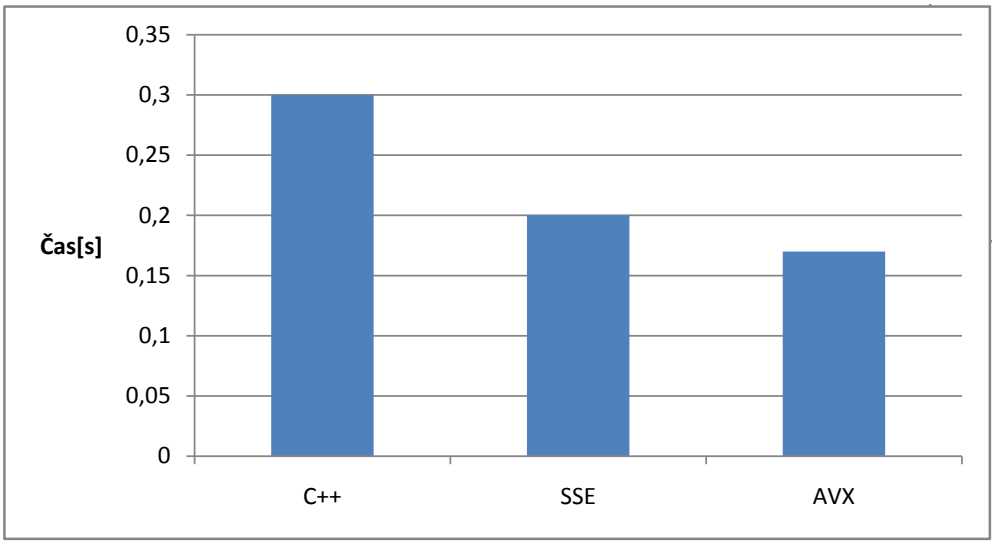
Pomocí instrukcí SIMD byly urychleny také další části metody Hog. Pro výpočet příznaků je nutné provést výpočet gradientů a následně vypočítat směr a velikost. Výpočet směru potřebuje dle vztahu (10) instrukci *atan2*, která není u visual c++ kompilátoru pro vektorové instrukce podporována. Proto bylo nutné převést implementaci atan2 z [21] na implementaci pomocí SIMD instrukcí. Implementace ze zdroje [21] poskytuje výpočet s chybou maximálně 0,01 radiánu, což ve výsledné aplikaci neudělalo žádný vedlejší efekt. Výhodou této implementace oproti klasické C++ verze z knihovny *math* byla také větší rychlost. Přibližně byla tato implementace rychlejší 1.5x než u klasické verze z C++. Další operace, která z důvodu vysokého počtu detekčních oken, byla časově velmi náročná je normalizace bloků, která je v grafu 43 také zahrnuta.



Obrázek 43: Graf urychlení výpočtu příznaků pomocí AVX a SSE instrukcí

Graf (43) zobrazuje výsledky pro získávání příznaků pomocí SSE a AVX instrukcí. Výsledky tohoto urychlená už nejsou tak značné jak u grafu (42). Důvodem takového zpomalení je vysoký počet operací, které provádějí naplnění registrů (operace _mm_set_ps) a operace zpětného kopírování z registrů do paměti (operace _mm_store_ps). V grafu jsou započítány také časy pro normalizaci jednotlivých bloků. U instrukcí AVX je velikost jednotlivých registrů 256 bitů a dokážou pojmout osm hodnot typu float. Pro velikost bloků, které nebyly násobkem osmi, musela být zbývající část pole provedena pomocí klasických instrukcí a vznikla tedy režie navíc, která výsledný čas zhoršila. U instrukcí díky velikosti pole, které bylo násobkem čtyř, nedocházelo k žádné režii navíc.

7.3.1 Shrnutí výsledků



Obrázek 44: Celkové urychlení výpočtu příznaků a klasifikace pomocí AVX a SSE instrukcí

Graf (45) porovnává výsledky urychlení pomocí různých instrukčních sad. Podle grafu (42), který zobrazuje výsledky urychlení operace skalárního součinu, jsou výsledky podle očekávaní u SSE a AVX 4x respektive 8x rychlejší než u klasických instrukcí. Tento příklad je však uměle vytvořen a nepřináší žádné problémy se kterýma se můžeme reálně setkat. Už u popisu grafu (43) je zmíněno několik důvodu, které vedly k mnohem horším výsledkům, než se očekávalo. Nutné je také zmínit složitost jednotlivých urychlených částí. Skalární součin (klasifikace), u kterého docházelo k nejlepším výsledkům urychlení, měl v rámci jednotlivých metod procentuálně nejmenší zastoupení, proto tato paralelizace nepřispěla k výraznému zrychlení. Metoda, která prováděla normalizaci jednotlivých bloků, měla v porovnání s klasifikací pomocí SVM o něco vetší procentuální zastoupení a tím také více přispěla k celému zrychlení aplikace. Porovnáním této metody z pohledu implementace pomocí instrukcí SSE a AVX ukázal hlavní důvod nepříliš velkého urychlení AVX oproti SSE. Problém nastal v častém volání této metody z důvodu časté normalizace relativně malých bloků a tudíž větší režii, která se projevila například při nulování registrů, kde toto nulování u AVX bylo dvakrát náročnější než u SSE. Největší procentuální zastoupení bylo u metody pro získávání příznaků, které ovšem z pohledu implementace vektorových instrukcí nepřineslo očekávané zrychlení kvůli častým využíváním instrukcí _mm_set_ps a _mm_store_ps.

8 Závěr

Cílem této práce bylo implementovat metodu Histogram orientovaných gradientů pro detekci rukou. Dále byla tato metoda rozšířena o využívání vektorových instrukcí pro zrychlení výsledné detekce objektů. Posledním bodem, který nebyl v zadání této práce, bylo implementovat metodu pro sledování nalezených objektů (rukou). V práci byly popsány parametry, které nejvíce ovlivňují výslednou detekci rukou a tyto parametry byly zahrnuty do výsledných testů. Testováním různých kombinací byla nalezena nejvhodnější kombinace pro detekci rukou. Tyto parametry byly také vyzkoušeny na implementaci, kterou poskytovala knihovna OpenCV a porovnány s vlastní implementací. Jelikož cílem této práce nebylo implementovat detektor, který dokáže rozpoznat ruku v jakémkoliv prostředí, ale pouze v omezeném prostoru, proto testovací snímky, které byly použity pro trénování, vznikly pouze v omezeném (jednom) prostředí. Jednou z hlavních nevýhod této techniky byla vysoká dimenze příznaků zkoumaného detekčního okna, která významným způsobem ovlivňovala výsledný čas detekce. Tato nevýhoda by se dala vyřešit změnou parametrů (změnou velikostí mřížky, bloků), ale výsledná detekce by s takovým nastavením nepřinesla uspokojivé výsledky. Další nevýhodou je nutnost testovat poměrné vysoké množství takových oken. Tento problém by se dal vyřešit vhodnou implementaci pro predikci umístění sledovaných objektů a tím by detekční ona mohla provádět skenování pouze na omezeném prostoru obrazu. Zmíněné nedostatky v rychlosti detekce měly být eliminovány použitím vektorových instrukcí. Použití těchto instrukcí však nepřineslo očekávané zrychlení. V implementaci bylo u některých metod dosaženo poměrně vysokého zrychlení, ale tyto části nepředstavovaly pro výsledný čas významné zpomalení jako u částí, kde se realizace využití vektorových instrukcí nedala tak snadno realizovat.

Jak už bylo uvedeno součástí práce, které sice nebylo uvedeno v zadání, bylo také rozšíření programu o sledování objektů (ruky). Na základě prostudovaných možností jednotlivých metod byl vybrán pro realizaci algoritmus Meanshift. Pro porovnání úspěšností této metody byl zvolen ještě algoritmus Camshift, který je rozšířením algoritmu Meanshift. Výsledné testování probíhalo na dvou videí z různých prostředí. Výsledky byly vyneseny do grafu, ve kterém byla porovnávána přesnost se skutečnou polohou objektu. Porovnáním algoritmů v kapitole experimenty lepší výsledky vykazoval algoritmus Camshift, ale za cenu poměrně vyšší časové náročnosti.

9 Reference

[1] "Applications and Clinical Benefits of CT Imaging." http://www.imaginis.com/ct-scan/applications-and-clinical-benefits-of-ct-imaging, 2010.

[2] "Detecting Defects in Dinnerware." http://www.vision-systems.com/articles/print/volume-16/issue-10/features/detecting-defects-in-dinnerware.html, 2011.

[3] "Surface bude podporovat Kinect." http://www.itbiz.cz/zpravicky/surface-bude-podporovat-kinect, 2012.

[4] "Asus Xtion Pro gets updated, Windows and Linux versions available." http://www.techfever.net/2011/07/asus-xtion-pro-gets-updated-windows-and-linux-versions-available/, 2011.

[5] "LEAP MOTION'S 3-D SYSTEM IS LIKE A KINECT FOR CONTROLLING YOUR COMPUTER." http://www.fastcompany.com/1837897/leap-motions-3-d-system-kinect-controlling-your-computer, 2012.

[6] "HAND." http://eofdreams.com/hand.html, 2013.

[7] Xiaojin Zhu, Jie Yang, A. Waibel, "Segmenting hands of arbitrary color." http://www.researchgate.net/publication/3845485_Segmenting_hands_of_arbitrary_color, 2000.

[8] Ying Wu , Qiong Liu , Thomas S. Huang, "An Adaptive Self-Organizing Color Segmentation Algorithm with Application to Robust Real-time Human Hand Localization." http://www.ifp.illinois.edu/yingwu/papers/ACCV00.pdf, 2000.

[9] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," pp. 511–518, 2001.

[10] Dalal N., Triggs B., *Computer Vision and Pattern Recognition*. IEEE Computer Society, 2005.

[11] Sojka Eduard, "Digitální zpracování a analýza obrazu." http://mrl.cs.vsb.cz/people/sojka/dzo/digitalni_zpracovani_obrazu.pdf, 2000.

[12] Nello Cristianini, John Shawe-Taylor, *Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[13] Souza, César R., "Kernel Functions for Machine Learning Applications.." http://crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html, 2010.

[14] Rosenhahn, Bodo ; Kersting, Uwe ; Andrew, Smith ; Brox, Thomas ; Klette, Reinhard ; Seidel, Hans-Peter, "A Silhouette Based Human Motion Tracking System." http://citr.auckland.ac.nz/techreports/2005/CITR-TR-164.pdf, 2005.

[15] Feng Wang, Xiangshi Ren, Zhen Liu, *Parallel and Distributed Processing with Applications*. 2008. 971-974.

[16] Chris Stauffer, W.E.L Grimson, "Adaptive background mixture models for real-time tracking." http://www.ai.mit.edu/projects/vsam/Publications/stauffer_cvpr98_track.pdf, 1999.

[17] Slobodan Ilić, "Mean Shift Tracking." http://campar.in.tum.de/twiki/pub/Chair/TeachingWs12TDCV/mean_shift.pdf, 2012.

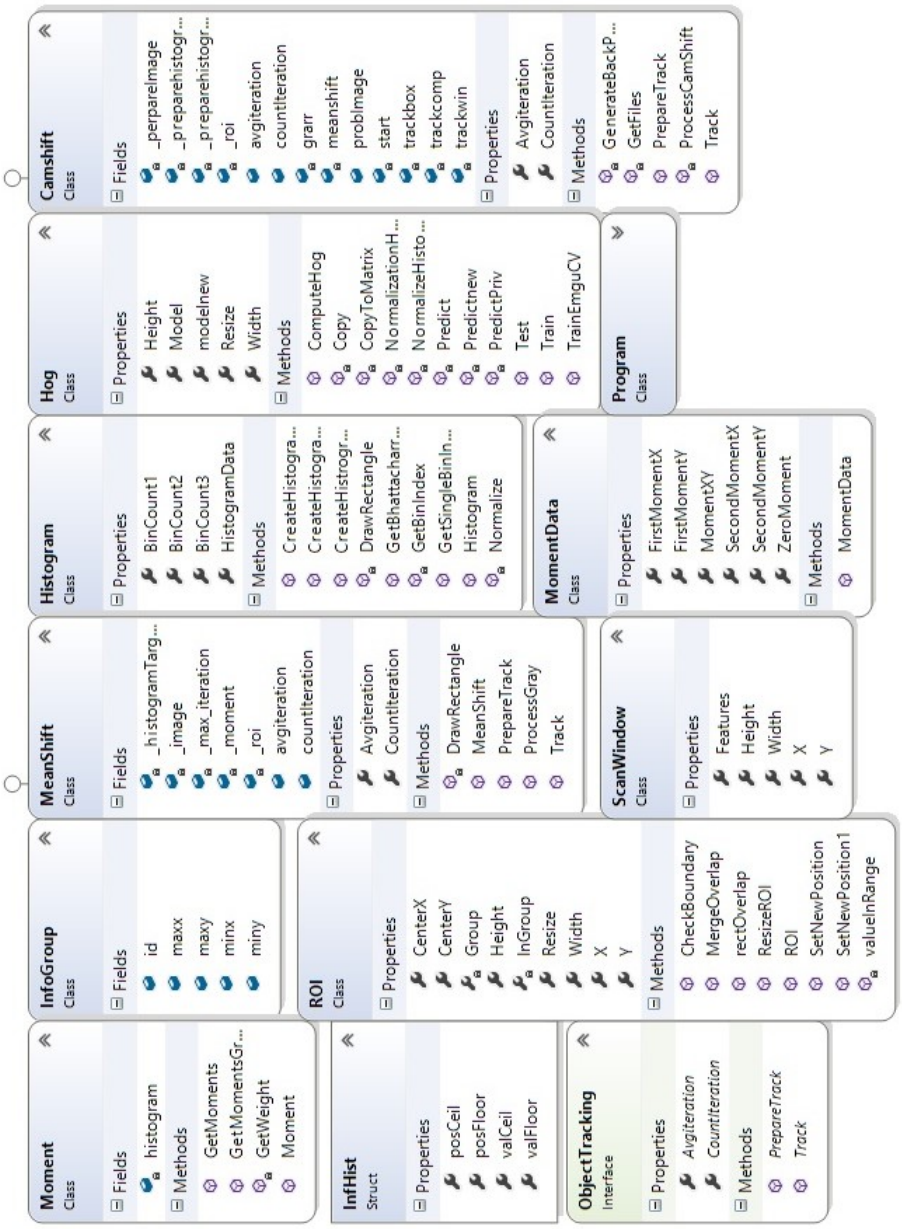
[18] Isaac Gerg, Adam Ickes, Jamie McCulloch, "CAMSHIFT Tracking Algorithm." http://www.gergltd.com/cse486/project5/, 2003.

[19] "Flynn's taxonomy." http://en.wikipedia.org/wiki/Flynn's_taxonomy, 2003.

[20] Pavel Tišnovský, "SIMD instrukce využívané v moderních mikroprocesorech řady x86." http://www.root.cz/clanky/simd-instrukce-vyuzivane-v-modernich-mikroprocesorech-rady-x86/, 2011.

[21] Jim Shima, "DSP Trick: Fixed-Point Atan2 With Self Normalization." http://dspguru.com/dsp/tricks/fixed-point-atan2-with-self-normalization, 1999.

A Diagram tříd



Obrázek 45: Diagram tříd implementace